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 than 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 tight to that of matrix multiplication via block recursive matrix factorizations [11]. 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 [6]. The classical approach is thus to reduce the computation modulo some primes and to recover the integer determinant from the modular computations. For this, at least to variants are possible: Chinese remaindering and p-adic lifting. The first variant requires either a good bound on the size of the determinant or an early termination probabilistic argument [7, §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 [3] to get an approximation of this determinant with a $O\left(n^3(\log(n)+\log(\|A\|))^2\right)$ bit complexity [14], , where $\|A\|$ is the largest entry in absolute value of the matrix. 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 the there exist integer matrices U, V with $\det U, \det V = \pm 1$, such that $A = USV$. s_i are called

*Université de Grenoble, laboratoire de modélisation et calcul, LMC-IMAG BP 53 X, 51 avenue des mathématiques, 38041 Grenoble, France. {Jean-Guillaume.Dumas, Anna.Urbanska}@imag.fr

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, approximate the determinant by p-adic lifting and recover only the remaining part (d/s_n) via Chinese remaindering. They where thus able to prove an expected complexity of $O(n^{\omega+1})$ bit operations.
Then G. Villard remarked that at most $O(\sqrt{n})$ invariant factors can be distinct and that, in general only the $O(\log(n))$ last ones are non trivial [10]. 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 [13]. 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.
In this paper, we propose a new way to extend the idea of [15, 16] to get the last consecutive invariant factors with high probability in section 3.2. Then we combine this with the scheme of [1] in an introspective way as explained in section 4. 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 [10]. 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 used with the very fast modular routines of [5, 6] and the LinBox library [4], 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 Chinese Remaindering loop (CRA) with early termination (ET) (see [8]), then proceed with LargestInvariantFactor algorithm to compute s_n (see [1, 10, 15]). [1]). We end the section with a sharpening of the result of [10] on the expected number of invariant factors.

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 $p_1 \cdots p_i$ via the Chinese reconstruction. The integer value of the determinant is thus computed as soon as the product of the p_i exceeds it. We know that the product is big enough if it exceeds some upper bound on the integer determinant 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 [8].

The following algorithm is an outline of a procedure to compute the determinant using CRA loop with early termination, correctly with probability $1-\epsilon$. If primes greater than l are randomly sampled from a set P ; if H is an upper bound for the determinant (e.g. Hadamard's bound: $|det(A)| \leq \sqrt{n||A||}^n$) and if r_n is the reconstructed result after n steps and if $det(A) \neq r_n$ then, at most $\log_l \frac{H-r_n}{p_0 p_1 \cdots p_{n-1}}$ distinct primes p_{n+1} would yield $r_{n+1} = r_n$ [8]. Thus, if r_i remains the same for k steps, either $det(A) = r_n$ or we have constantly chosen bad primes. This happens only in that proportion:

$$\left(\frac{\log_l \frac{H-r_n}{p_0 p_1 \cdots p_{n-1}}}{|P|} \right)^k . \tag{1}$$

Therefore, the probabilistic early terminated Chinese remainder determinant is as follows:

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: $H = \sqrt{n||A||}^n$; // Hadamard's bound
2: **repeat**
3: Get a prime p_i from the set P;
4: Compute result $q_i \mod p_i$; //via LU factorization of A modulo p_i.
5: Reconstruct r_i, the determinant modulo $p_1 \cdots p_i$; // by Chinese remaindering
6: $k = \log(\frac{\log_l \frac{H-r_i}{p_0 p_1 \cdots p_i}}{|P|}) / \log(\epsilon)$;
7: **until** $r_{i-k} = \cdots = r_i$ or $\prod p_i > H$

To compute the modular determinant in step 2.1 we use LU factorization algorithm and we refer to it as LU iteration. Early termination is particularly useful in the case when the computed value is much smaller than the a priori bound. Therefore 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 [14] and later in the form of the *LargestInvariantFactor* procedure (LIF) in [1, 10, 8, 15]. 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 for 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 [3], enables us to get s_n with high probability at cost of $O(n^3(\log(n) + \log(\|A\|))^2)$ independent of the size of s_n.

The following algorithm takes as input parameters B and r which are used to control the probability of correctness. r is the number of successive solvings and B the size of the random set from which values of the random vector b are chosen.

Algorithm 2.2 LIF

Require: An integer $n \times n$ matrix A.
Require: A stream S_β of random integers uniformly chosen from the set $\{0, 1 \ldots, \beta - 1\}$.
Require: A number of iterations $r \leq 1$.
Ensure: $\tilde{s}_n$, a factor of $s_n(A)$.
$\tilde{s}_n$ equals $s_n(A)$ with probability depending on r and β given by Theorem 2.1

1: $\tilde{s}_n = 1$;
2: **for** $i = 1$ to r **do**
3: Generate b_i a random vector of dimension n for the stream S_β;
4: Solve $Ax_i = b_i$ over the rationals using Dixon lifting;
5: $d :=$ lcm(denominators of entries of x_i);
6: $\tilde{s}_n = \text{lcm}(\tilde{s}_n, d)$;
7: **end for**
8: Return: $\tilde{s}_n$.

The following theorem characterizes the probabilistic behavior of the LIF procedure.

Theorem 2.1. *Let A be a $n \times n$ matrix, H its Hadamard's bound. The output $\tilde{s}_n$ of Algorithm 2.2 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 $r = 2$, $\beta = \lceil (n+1)H \rceil$ then $s_n = \tilde{s}_n$ with probability at least 2/3;

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 $\beta \geqslant 3$ then $s_n(A) = \tilde{s}_n$ with probability at least $1 - \epsilon$;

Proof. The proofs of (i), (ii) and (iv) are in to [1]. The proof of (iii) is in [10]. 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

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

□

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

Now, the idea of [1] is that one can combine both the Chinese remainder and the LIF approach. Indeed, one could compute first s_n and then reconstruct only the remaining factors of the determinant d/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 [15, 16] 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 $3log_\lambda(n) + 32$ in general if the entries of the matrix are chosen in a set of λ consecutive integers [10]. As they also give a way to compute any $s_i(A)$ this gives 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 $O(\log_\lambda)$ invariant factors of A slightly faster than by [10]. 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 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 [10] 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 $\log(n)$ last invariant factors

3.1 On the number of invariant factors

The expected performance of our algorithm depends strongly on the number of non trivial invariant factors of A. If there is only one invariant factor, as it seems to be the case for many matrices, then the algorithm runs in approximately the time of solving systems. The sign of the determinant can then quickly be determined by a few CRA loop iterations. The performance of early termination being especially outstanding as $\det(A)/K$ is in general several times smaller than H/K.
The result in [10] 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 $3\log(n) + 32$. In search for more exact 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* $\log_\lambda(n) + \log_\lambda(\log_\lambda(n) + 2) + 9$.

Both proofs can be found in the appendix. Notice, that in the average case our improvement allows us to consider only primes less than λ in the case where $\lambda > n$.

3.2 Extended Bonus Idea

With our estimation of the expected number of invariant factors, we may assume that we do not have to compute more than $\log_\lambda(n) + \log_\lambda(\log_\lambda(n) + 2) + 9$ of those. If it turns out that the CRA loop does not stop at this point, we can switch to another algorithm to achieve a better worst case complexity.
In his thesis [16], 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 refereed as bonus. Here, we extend this idea to the computation of the $(n-k)$th factor with about $(k+1)$ solvings.
Suppose $n^{(j)}$, $j = 1, 2 \ldots (k+1)$ is the vector of numerators of the rational solution $x^{(j)}$ of the equation $Ax^{(j)} = b^{(j)}$, where $b^{(j)}$ is a random vector. The $x^{(j)}$ have a common denominator $\tilde{s}_n$. Let B denote the $n \times (k+1)$ matrix $[b^{(j)}]_{j=1,\ldots,k+1}$. Following Wan, we notice that $s_n(A)A^{-1}$ is an integer matrix, the Smith form of which is equal to

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

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 by right multiplying it by B. Indeed, $s_n(A)A^{-1}B = [x^{(j)}]_{j=1,\ldots,k+1}$ is already computed once $k+1$ systems have been solved and $s_{k+1}(s_n(A)A^{-1}B)$ is a multiple of $s_{k+1}(s_n(A)A^{-1})$. Obtaining this multiple from the solution vectors, is then only to perform several $(k+1) \times (k+1)$ determinants. We detail this in the following theorem:

Theorem 3.3. *Let M be a $n \times n$ integer matrix. Let R_i be a random integer $n \times i$ matrices, $n > i$, with entries in $\{0, 1 \ldots, \beta - 1\}$. Then the greatest common divisor of μ independent $i \times i$ minors of MR_i is equal to $s_{n-i}(M)$ with probability at most $1 - \frac{1}{2^{\mu-1}}$ as soon as $\mu \geqslant 2$. Moreover, if $\mu > 2$, the expected value of $s_n(MR_i)/s_nM$ is 1.*

Proof. We first need to prove that $s_i(M)|s_i(MR_i)$ for a random $n \times i$ matrix R_i. Consider the Smith form of MR_i modulo $s_i(M)$. As the modular rank of MR_i is less than i, $s_i(MR_i)$ mod $s_i(M) = 0$ as required, see [16, 15]. This property holds also for a product L_iM as well as for the product of three matrices L_iMR_i. However, for MR_i we may additionally use the following argumentation.
Notice that the Smith form of the product matrix MR_i is equivalent to the product of the Smith forms of M and R_i. Therefore it is equal to that of M provided R_i has a trivial Smith form. This is very likely to happen when R_i is highly rectangular. It suffices that $d_i(R_i) = 1$, where d_i is the gcd of all $i \times i$ minors. As the matrix R_i is chosen randomly, there are at least $\lfloor \frac{n}{i} \rfloor$ independent minors. Now for a given prime p, the probability that p divides one of the $i \times i$ minor of R_i is $\frac{1}{p}$, thus the probability that p divides μ independent

minors is $\frac{1}{p^\mu}$. Therefore the probability that their gcd is non-trivial is the sum of these probabilities over all possible primes. Now the $i \times i$ minors of R_i are bounded by $\sqrt{\beta i}^i$, by Hadamard. The overall probability is then

$$\sum_{prime\ p,\ p<\sqrt{\beta i}^i} \left(\frac{1}{p}\right)^\mu < \zeta_p(\mu)$$

where ζ_p is the prime zeta function (the sum over all the primes). For instance, $\zeta_p(2)$ is 0.452247 and $\zeta_p(9)$ is 0.000993604. Trivially, $\frac{1}{2^\mu} < \zeta_p(\mu)$ and bounding the other terms by $\int_2^\infty \frac{1}{x^\mu} dx$ gives $\zeta_p < \frac{1}{2^\mu} + \frac{1}{(\mu-1)2^{\mu-1}} < \frac{1}{2^{\mu-1}}$ for $\mu > 2$. For $\mu = 2$, $0.452247 < 0.5$ and the bound is also correct.
The expected size of $s_n(MR_i)/s_n(M)$ can be calculated as the sum over primes p

$$\sum_{p<\sqrt{(\beta i)^i}} \sum_{l=1}^{\infty} \log(p^l) \left(\frac{1}{p^l}\right)^\mu \leqslant \sum_{p<\sqrt{(\beta i)^i}} \sum_{l=1}^{\infty} \left(\frac{1}{p^l}\right)^{\mu-1} = \sum_{p<\sqrt{(\beta i)^i}} \frac{1}{p^{\mu-1}-1}$$

For $\mu > 2$ this value is at most $2\zeta_p(\mu-1)$, which is less than 1.

□

3.3 Last invariant factors

Using the analysis of the previous section we remark that the number k of non-trivial invariant factors is small in general. The size of the entries in $M = s_n(A)A^{-1}$ can however be as large as $O(n(\log(n) + \log(\|A\|)))$. We therefore propose an algorithm minimizing the effect of the size of the entries, and do not try to minimize the effect of k. Moreover, to get a good probability of success, we will require that $k \leqslant \frac{-1+\sqrt{1+4n}}{2}$. If $k = O(log(n))$, this is easily guaranteed.
First, notice that $s_i = d_i/d_{i-1}$. We can therefore attempt to calculate d_i using some minors of MR_i. As before, the gcd of j minors, can differ from d_i with probability less than $\zeta_p(j) \leqslant 2^{-j+1}$. To obtain k factors of M we can compute them as a sequence so that d_{i-1} is already computed at step i. The first step is to compute the gcd of all the entries of the matrix. The algorithm that computes k smallest invariant factors of A with probability at least $1-\epsilon$ as soon as for $k \leqslant \frac{-1+\sqrt{1+4n}}{2}$ is as follows.

Theorem 3.4. *If $k \leqslant \frac{-1+\sqrt{1+4n}}{2}$, algorithm 3.1 correctly computes $s_1(A), \ldots s_k(A)$ with probability at least $1-\epsilon$.*

Proof. All k factors are correct if all d_i are correct. We recognize two cases in which d_i is overestimated. First, an unlucky matrix R_i might have been chosen and second, the number of minors in step 9 can be insufficient. As we repeat our choice N times, the probability of this is at most $2P(N\mu_i)$ in step i. The overall probability of error for k factors is now

$$\sum_{i=2}^{k} 2P(N\mu_i) < \sum_{i=2}^{k} 2^{-N\mu_i+2}.$$

Algorithm 3.1 k-LastInvariantFactors

Require: An integer $n \times n$ matrix $M = (a_i j)_{i,j=1..n}$.
Require: A stream S_β of random integers uniformly chosen from the set $\{0, 1 \dots, \beta - 1\}$.
Require: A number $k \leqslant \frac{-1+\sqrt{1+4n}}{2}$ of factors to compute.
Require: $0 < \epsilon < 1$
Ensure: $\tilde{s}_1, \dots, \tilde{s}_k$, multiples of $s_1(M), \dots s_k(M)$.

$\tilde{s}_1 = d_1 = \gcd(a_{ij} : i, j = 1 \dots n)$;
$N = \frac{\lceil \log(8/\epsilon) \rceil}{\lfloor n/k \rfloor}$;
for $i = 2$ to k **do**
  $\mu_i = \lfloor \frac{n}{i} \rfloor$;
  $d_i = 0$;
  for $t = 1$ to N **do**
    generate a random $n \times i$ matrix R_i;
    calculate MR_i;
    choose μ_i distinct $i \times i$ submatrices of MR_i and calculate minors $m_j, j = 1, \dots \mu_i$;
    $d_i = \gcd_{j=1..\mu_i}(d_i, m_j)$;
  end for
  $s_i = \frac{d_i}{d_{i-1}}$;
end for

For every m there are at most $(\frac{n}{m} - \frac{n}{m+1})$ i such that $\mu_i = \lfloor \frac{n}{i} \rfloor = m$. We may therefore estimate

$$\sum_{m=\mu_k}^{\mu_2} \frac{n}{m(m+1)} 2^{-Nm+2} < 2^{-N\mu_k+3} \frac{n}{\mu_k(\mu_k+1)} = 2^{-N\mu_k+3} \frac{n}{\lfloor \frac{n}{k} \rfloor (\lfloor \frac{n}{k} \rfloor + 1)} \leqslant 2^{-N\mu_k+3} \frac{k^2}{n-k}.$$

We force k to be less than $\frac{-1+\sqrt{1+4n}}{2}$ so that $\frac{k^2}{n-k} \leqslant 1$. Then, $N = \frac{\lceil \log(8/\epsilon) \rceil}{\lfloor n/k \rfloor}$ is chosen so that $2^{-N\mu_k+3} \leqslant \epsilon$. □

Theorem 3.5. *The complexity of computing first k factors of A, $k \leqslant \frac{-1+\sqrt{1+4n}}{2}$ by algorithm 3.1 with probability at least $1-\epsilon$ is $O\left(\log(16/\epsilon)(n^2 k \log(\|M\|) + nk^4(\log(k) + \log(\|M\|)))\right)$.*

Proof.
The cost of all matrix multiplications is $\sum_{i=2}^{k} N(n^2 i) \log(\|M\|)$ which is less than $Nk(k+1)n^2 \log(\|M\|)$. For the choice of N and k we have $Nk \leqslant \lceil \log(8/\epsilon) \rceil \leqslant \log(16/\epsilon)$. The cost of matrix products is therefore $O\left(\log(16/\epsilon) n^2 k (\log \|M\|)\right)$. The overall cost of minors computation can be bounded by the cost of calculating n $k \times k$ minors kN times. As $k \times k$ determinant can be computed in $O(k^4(\log(k) + \log(\|M\|)))$ time (for simplicity we considered the complexity of CRA determinant algorithm here) we get $O\left(\log(16/\epsilon) n k^4 (\log(k) + \log(\|M\|))\right)$ and the overall complexity is as required. □

4 Introspective Algorithm

Now we should incorporate algorithms 2.1, 2.2 and 3.1 in a form of an introspective algorithm. CRA loop refers here to algorithm 2.1, slightly modified to compute $det(A)/K$. If we re-run CRA loop, we use modular determinant results already computed to recover $det(A)/K \mod p$. Notice that the loop of algorithm 3.1 has been split so that solutions obtained by system solving can be used at each step, instead of another random matrix R_i.

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

Proof. Termination is possible only by early termination of the CRA loop or by the determinant algorithm used in the last step. In both cases $1 - \epsilon$ probability is ensured. □

4.1 Complexity

The following theorem gives the complexity of the algorithm.

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

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

where $\mathrm{O}^{\sim}$ *hides some* $\log(\log(n))$ *factors. The pessimistic complexity is that of 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 above, the complexity of initial CRA iteration is $\mathrm{O}^{\sim}\left(n^{\omega}\log(1/\epsilon)\right)$. The loop will iterate for at most $(i_{max}+1)N$ iterations, giving the complexity of system solving equal $O\left((i_{max}+1)Nn^3(\log(n)+\log\|A\|)^2\right)$. It is the same for the CRA loop used later, because of the time limit. The choice of N ensures a probability $1-\delta$ (here we set $\delta = n^{-1}$) of computing exactly the $(i_max + 1)$ invariant factors of $s_n(A)A^{-1}$. This results with a complexity $O\left(\log(16n)ni_{max}^4(\log(i_{max}) + n\log(n))\right)$ for this task. To resume the CRA loop we only perform modular division and reconstruct the result using Chinese remaindering. The cost of this step is thus negligible when compared with the others. For the last step we propose the $O^{\sim}(n^{3.2}\log(\|A\|))$ algorithm of Kaltofen [13] and refer to [12] for a survey on complexity of determinant algorithms.
As the expected number of invariant factors is equal to i_max, we may suspect that the algorithm will not reach the last step and thus the worst case complexity. The choice of N tries to ensure that the under-estimation is a small constant and thus can be recovered by the CRA loop steps. However, we must still assume in this case, that all factors were computed without over-estimation. Careful examination yields that with probability at least $1-\delta$ the complexity of the algorithm is $Nn^3(\log(n)+\log\|A\|)^2\sum_{i=1}^{i_{max}} iP(\#factors = i) + n^{3.2}P(\#factors > i_max)$ which can be evaluated as $Nn^3(\log(n)+\log\|A\|)^2\mathbf{E}(\#factors) + n^{3.2}P(\#factors > i_max)$. With probability at most δ the algorithm still runs with $\mathrm{O}^{\sim}(n^{3.2})$ complexity. Summarizing, we may notice, that the when $\mathbf{E}(\#factors) < i_{max}$ the algorithm runs in the $\mathrm{O}^{\sim}\left(\log(16/\delta)n^3(\log(n)+\log\|A\|)^2\right)$ time with probability at least $1-\delta$ and

Algorithm 4.1 Extended Bonus Algorithm

Require: An integer $n \times n$ matrix A.
Require: $0 < \epsilon < 1$, an error tolerance.
Require: A stream S_β of random integers uniformly chosen from the set $\{0, 1 \ldots, \beta - 1\}$.
Require: A set P of random primes greater than l.
Ensure: The integer determinant of A, correct with probability at least $1 - \epsilon$.

$k = \log(\frac{\log_l(H)}{|P|})/\log(\epsilon)$; //H - Hadamard's bound for A
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
$K = 1; j = 0; i = 0$;
$i_{max} = \min\{2(\log_\lambda(n) + \log_\lambda(\log_\lambda(n) + 2) + 9), \frac{n}{3}, \frac{-1+\sqrt{1+4n}}{2}\}$; //the expected number of nontrivial factors
$N = \frac{\lceil \log(8n) \rceil}{\lfloor n/i_{max} \rfloor}$;
while $i < i_{max}$ **do**
  Generate b_j a random vector of dimension n from the stream S_β; j=j+1;
  Compute $\tilde{s}_n$ by solving $Ax_j = b_j$; //see Alg. 2.2
  if $i = 0$ **then**
    $i = 1$
  else if $(i + 1)N \leqslant j$ **then**
    compute $\tilde{s}_{n-i} = \tilde{s}_n / \tilde{s}_{i+1}(s_n(A)A^{-1})$ using $(i + 1)$th step of Alg. 3.1 and matrices $[b_1 \ldots b_{N(i+1)}]$
    i=i+1;
  end if
  $K = \tilde{s}_n \cdot \cdots \cdot \tilde{s}_{n-i}$;
  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 $\tilde{s}_{n-i} = 1$ **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 K;
    else
      $i = i_{max}$;
    end if
  end if
end while
run an asymptotically better integer determinant algorithm;

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

Figure 1: Comparison of the performance of Algorithms 4.1 with i_{max} set to 1 and 2 on engineered matrices.

in $\mathrm{O}^{\sim}(n^{3.2})$ time with probability at most $P(\#factors > i_max) + \delta$. Setting $i_max = 2\mathbf{E}(\#factors)$ forces $P(\#factors > i_max)$ to be $O(n^{-1})$ (see (4)) and the choice of $\delta = \frac{1}{n}$ suffices to say that the expected complexity is $\mathrm{O}^{\sim}\left(\log(16n)n^3(\log(n) + \log\|A\|)^2\right)$. □

5 Experiments and Further Adaptivity

The described algorithm was implemented in the LinBox exact linear algebra library [4]. In a preliminary version i_max was set to 2 or 1 and the switch in the last step was not implemented. This was 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 specially engineered matrices which have the same Smith form as $diag\{1, 2, .., n\}$ and the number of invariant factors of about $\frac{n}{2}$.

For this matrices, 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} started to pay out, but we did not yet attempt to compute the next (though still nontrivial) factor.

The switch between winners can be explained by the fact that in some situations, obtaining factor s_{n-i} by LU-factorization (which costs $\frac{\log(s_{n-i})}{l}$ 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 $\log(\tilde{s}_{n-i}) = 1$ to

$$\log(\tilde{s}_{n-i}) \leqslant \frac{time(solving)}{time(LU)} \log(l),$$

if the primes used in the CRA loop are greater that 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*,

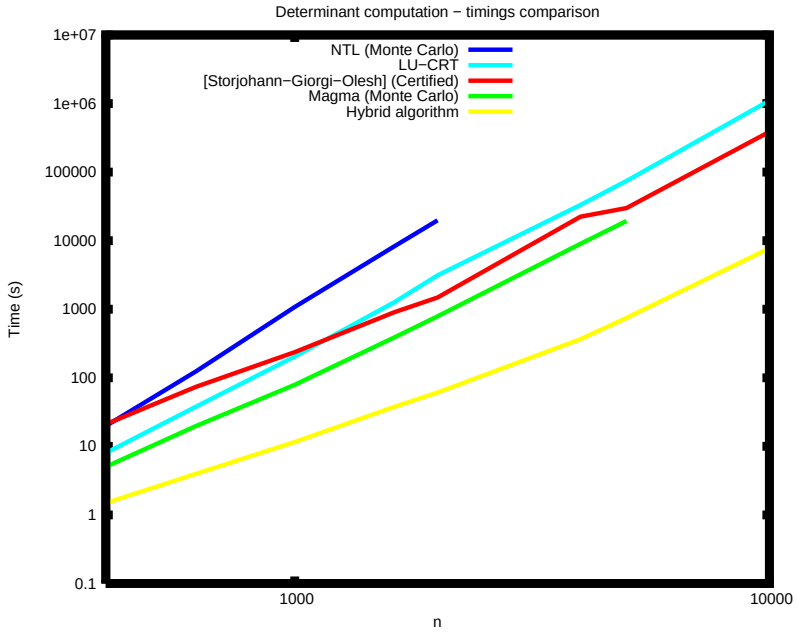

Figure 2: Comparison of our algorithm with other existing implementation. Tested on random dense matrices of the order 400 to 10000, with entries {-8,-7,. . . ,7,8} Using fast modular routines puts our algorithm several times ahead of the others. Scaling is logarithmic.

this condition could be verified before its calculation. Results like Eq. (2) can be helpful here.
For a generic case of random dense matrices another observation was that the bound for the number of invariant factors is then quite crude. Indeed for random matrices, the algorithm nearly always stopped with early termination after one system solving. This together with fast underlying arithmetics accounted for the superiority of our algorithm is as seen in figure 5 where comparison of timings for different algorithms is presented. A modification to be tested, could be to try to reconstruct s_n with only one entry of the solution vector $x = \mathbf{v}/d$.

6 Conclusions

In this paper we present an algorithm computing the determinant of an integer matrix which expected time complexity is $O\left(n^3(\log(n) + \log(||A||))^2 \log(n)\right)$. I in fact be closer to $O\left(n^3(\log(n) + \log(||A||))^2 k\right)$ if the number k of non-trivial invariant factors is smaller than *a priori* expected. 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 non-trivial 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 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]
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 to perform several p-adic liftings in parallel, but with less iterations [9]. There the issue is to perform an optimal distributed early termination.

References

[1] John Abbott, Manuel Bronstein, and Thom Mulders. Fast deterministic computation of determinants of dense matrices. In Sam Dooley, editor, *ISSAC 99: July 29–31, 1999, Simon Fraser University, Vancouver, BC, Canada: proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, pages 197–204, 1999.

[2] Zhuliang Chen and Arne Storjohann. A BLAS based C library for exact linear algebra on integer matrices. In Manuel Kauers, editor, *ISSAC '05: July 24–27, 2005, Beijing, China: Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation*, pages 92–99, pub-ACM:adr, 2005. pub-ACM.

[3] John D. Dixon. Exact solution of linear equations using p-adic expansions. *Numerische Mathematik*, 40:137–141, 1982.

[4] Jean-Guillaume Dumas, Thierry Gautier, Mark Giesbrecht, Pascal Giorgi, Bradford Hovinen, Erich Kaltofen, B. David Saunders, Will J. Turner, and Gilles Villard. LinBox: A generic library for exact linear algebra. In Arjeh M. Cohen, Xiao-Shan Gao, and Nobuki Takayama, editors, *Proceedings of the 2002 International Congress of Mathematical Software, Beijing, China*, pages 40–50. World Scientific Pub, August 2002.

[5] Jean-Guillaume Dumas, Thierry Gautier, and Clément Pernet. Finite field linear algebra subroutines. In Teo Mora, editor, *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, Lille, France*, pages 63–74. ACM Press, New York, July 2002.

[6] Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. FFPACK: Finite field linear algebra package. In Jaime Gutierrez, editor, *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, Santander, Spain*, pages 63–74. ACM Press, New York, July 2004.

[7] Jean-Guillaume Dumas, Clément Pernet, and Zhendong Wan. Efficient computation of the characteristic polynomial. In Manuel Kauers, editor, *Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation, Beijing, China*, pages 140–147. ACM Press, New York, July 2005.

[8] Jean-Guillaume Dumas, B. David Saunders, and Gilles Villard. On efficient sparse integer matrix Smith normal form computations. *Journal of Symbolic Computations*, 32(1/2):71–99, July–August 2001.

[9] Jean-Guillaume Dumas, Will J. Turner, and Zhendong Wan. Exact solution to large sparse integer linear systems, May 2002. East Coast Computer Algebra Day. Long Island City, New York, USA.

[10] Wayne Eberly, Mark Giesbrecht, and Gilles Villard. Computing the determinant and Smith form of an integer matrix. In *Proceedings of The* 41^{st} *Annual IEEE Symposium on Foundations of Computer Science, Redondo Beach, California*, November 2000.

[11] Oscar H. Ibarra, Shlomo Moran, and Roger Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45–56, March 1982.

[12] Erich Kaltofen and Gilles Villard. Computing the sign or the value of the determinant of an integer matrix, a complexity survey. *Journal of Computational and Applied Mathematics*, 164:133–146, 2004.

[13] Erich Kaltofen and Gilles Villard. On the complexity of computing determinants. *Computational Complexity*, 13(3-4):91–130, 2005.

[14] Victor Pan. Computing the determinant and the characteristic polynomial of a matrix via solving linear systems of equations. *j-INFO-PROC-LETT*, 28(2):71–75, June 1988.

[15] David Saunders and Zhendong Wan. Smith Normal Form of dense integer matrices fast algorithms into practice. In Jaime Gutierrez, editor, *ISAAC 2004: July 4–7, 2004, University of Cantabria, Santander, Spain: proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, pages 274–281, 2004.

[16] Z. Wan. *Computing the Smith Forms of Integer Matrices and Solving Related Problems*. PhD thesis, University of Delaware, USA., July 2005.

Appendix

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

Lemma. 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:

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

□

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

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

Proof.[Theorem 3.1] The idea of the proof similar is to that in [10]. Let A be a random matrix with entries in the set $\{0, 1, 2 \ldots \lambda - 1\}$.
Let $MDep_i(p)$ denote an event that the submatrix A_i, including first i columns of $A \mod p$ has rank at most $i-2$ over $\mathbf{Z}_p$. Notice, that the event $MDep_i(p)$ can occur only if p divides one of the $(i-1) \times (i-1)$ minors of A_{i-1}.
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 exist a set of $(i-2)$ rows R_{i-2} which has full rank. Consider 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} with probability λ^{-1}. For $p < \lambda$ this probability is $\frac{1}{\lambda}\lceil\frac{\lambda}{p}\rceil$ which is always greater than $\frac{2}{p+1}$. 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 be $(\frac{2}{p+1})^{n-i+2}\frac{p+1}{p-1}$ 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. $A \mod p$ has then rank at most $n-j$ over $\mathbf{Z}_p$. This in consequence means that for $j > 1$ submatrix A_{n-j+2} has rank at most $n-j$, so the event $MDep_i(p)$ is fulfilled. Therefore matrix A has at least j invariant factors divided by p with probability

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

Now the expected number of invariant factor divided by p is

$$3 + 3\sum_{j=3}^{j=n}(\frac{2}{3})^j = 3 + 9(\frac{2}{3})^3 \leqslant 6, \quad p = 2,$$

$$1 + \sum_{j=1}^{j=n}(\frac{2}{p+1})^j \frac{p+1}{p-1} = 1 + \frac{2(p+1)}{(p-1)^2} \leqslant 3, \quad 2 < p < \lambda,$$

$$1 + \frac{\lambda}{(\lambda-1)^2} < 2, \quad p \geqslant \lambda > 2. \tag{3}$$

□

Proof.[Theorem 3.2] In addition to $MDep_i(p)$ introduced earlier, let Dep_i denote an event that first i columns of A are linearly independent and $MDep_i$, an event that either of $MDep_i(p)$, p-prime, 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.
or primes $p \geqslant \lambda$ we should estimate the number of primes dividing the $(i-1)$th minor. By Hadamard's bound (notice that Dep_{i-1} does not hold), the minors are bounded in absolute value by $((i-1)\lambda^2)^{i-1}$. Therefore the number of primes $p \geqslant \lambda$ dividing the minor is at most $\frac{i-1}{2}(\log_\lambda(i-1)+2))$. Summarizing,

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

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

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

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 $n[\log_\lambda(n)+2] \leqslant \lambda^j$, and therefore $\log_\lambda(n)+\log_\lambda(\log_\lambda(n)+2) \leqslant j$. Consequently, the expected number of invariant factors is

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

$$+ 2(\frac{1}{2})^j + \frac{n-j+1}{2}(\log_\lambda(n-j+1)+2))(\frac{1}{\lambda})^j\frac{\lambda}{\lambda-1}\big) = \lceil \log_\lambda(n)+\log_\lambda(\log_\lambda(n)+2)\rceil +$$

$$+ 1 + \frac{\lambda+1}{(\lambda-1)^2} + 4 + 1 \leqslant \log_\lambda(n)+\log_\lambda(\log_\lambda(n)+2)+9.$$

□