HAL
open science

Towards an exact adaptive algorithm for the determinant of a rational matrix

Anna Urbanska

To cite this version:

Anna Urbanska. Towards an exact adaptive algorithm for the determinant of a rational matrix. 2007. hal-00150872

HAL Id: hal-00150872

https://hal.science/hal-00150872v1

Preprint submitted on 31 May 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards an exact adaptive algorithm for the determinant of a rational matrix

Anna Urbańska
Laboratoire Jean Kuntzmann
Université Joseph Fourier, Grenoble I
E.mail: Anna.Urbanska@imag.fr

hal-00150872, version 1 - 30 May 2007

ABSTRACT

In this paper we propose several strategies for the exact computation of the determinant of a rational matrix. First, we use the Chinese Remaindering Theorem and the rational reconstruction to recover the rational determinant from its modular images. Then we show a preconditioning for the determinant which allows us to skip the rational reconstruction process and reconstruct an integer result. We compare those approaches with matrix preconditioning which allow us to treat integer instead of rational matrices. This allows us to introduce integer determinant algorithms to the rational determinant problem. In particular, we discuss the applicability of the adaptive determinant algorithm of [9] and compare it with the integer Chinese Remaindering scheme. We present an analysis of the complexity of the strategies and evaluate their experimental performance on numerous examples. This experience allows us to develop an adaptive strategy which would choose the best solution at the run time, depending on matrix properties. All strategies have been implemented in LinBox linear algebra library.

1. INTRODUCTION

The determinant computation is one of the core problems in linear algebra. To our knowledge, the problem of the exact computation of the determinant of a rational matrix (i.e a matrix with rational entries) has not yet been widely studied. In general, exact algorithms can be used everywhere where large precision is required. For example, the determinant can be too close to 0 or $\pm\infty$ and thus cannot be computed by floating point precision algorithms. In the case of ill-conditioned matrices symbolic methods can be preferred as rounding errors can spoil the computation. It can also be interesting to compare the use of decimal and continued fractions approximations of the entries of real-valued matrices. Continued fractions are the best approximants with small denominators, see [12, Ch. 4]. In this paper, we will try to face the question of how efficient an exact determinant computation can be in both cases.

LinBox library [7] implements exact algorithms for the determinant computation in the case of modular and integer domains. By using fast modular routines [6, 8] it can offer solutions an order of magnitude faster than other existing implementations [9]. We apply these procedures to the computation of the determinant of a rational matrix.

Rational field arithmetics is implemented in GMP [21] and Givaro [22] libraries. In general, rational numbers are difficult to treat from the exact computation point of view. Mainly, the size of the numerator and denominator can increase very quickly with every addition and multiplication. When we add or multiply two fractions with numerators and denominators bounded by M, the numerator and denominator of the result are bounded by $O(M^2)$. Moreover, one addition requires 3, and one multiplication requires 2 integer products, as well as a gcd computation. Therefore, the cost of an exact matrix-vector or matrix-matrix product can be prohibitive in practice. This prohibits the use of the rational field $\mathbb{Q}$ in most exact linear algebra algorithms which rely on matrix-matrix or matrix-vector products.

However, the cost of computing a modular image of a rational number $\frac{a}{b}$, where a, b are of moderate size, should be comparable with the cost of computing a modular image of a large integer number. This allows us to compute a modular image of a rational matrix at a reasonable cost and thus enables us to use modular procedures.

To compute the determinant of a rational matrix $A = \left[\frac{a_{ij}}{b_{ij}}\right], b_{ij} > 0$ the problem of matrix storage has to be considered. First, we can store the entries of A as rational numbers. Furthermore, one could store the common denominator $D(A)$ of all entries of A and an integer matrix A' given by the formula $A = \frac{1}{D(A)}A'$. This approach can be useful in the case when the entries of A are decimal fractions and $D(A)$ can be set to a power of 10. But if we only assume that the values $|a_{ij}|, b_{ij}$ are less than M, both $D(A)$ and $\|A'\|$ are bounded by $O(M^{n^2})$. Still, we may store the common denominator for each row (column) separately. Then the integer vectors $\tilde{A}_i$ are given by the equation $A_i = \frac{1}{D_i}\tilde{A}_i$, where A_i is the matrix row (column) and D_i is the common denominator of its entries. Vectors $\tilde{A}_i$ form matrix $\tilde{A}$, the norm of which is bounded by $O(M^n)$. The product πD_i gives a more accurate approximation of the denominator $D(\det(A))$ than $D(A)^n$.

The purpose of this paper is to propose the strategies to compute the denominator of a rational matrix. All approaches are based on modular computation. Depending on the matrix storage determinant and/or matrix preconditioning is proposed. The resulting algorithms can use the rational reconstruction [12, Ch.5] and/or existing integer determinant algorithms.

The rest of the paper is organized as follows. In section 2 we give a short description of the existing algorithms for the rational reconstruction and the integer determinant problem. In section 3 we present the main result i.e. two preconditioning strategies and four new algorithms to compute the rational determinant. The cost of the algorithms can be described in terms of the number of modular images of A and modular determinant computations needed. Depending on the strategy, the cost of the rational reconstruction or p-adic lifting is taken into account. In section 4 we discuss the cost of computing a modular image of a matrix and the overall cost of the algorithms. In section 5 we present the experimental results and discuss the best choice of the strategy in practice. We conclude the paper by proposing some mixed solutions in section 6.

2. EXISTING ALGORITHMS

The aim of this section is to introduce the algorithms that will be used later in section 3. In subsection 2.1 we give a short description of the rational reconstruction procedure. On the example of p-adic system solving [2], we present the application of this procedure to the computation of a rational solution. We show how to change the procedure in the case of early terminated reconstruction [18] and give the complexity estimation in this case. Then in subsection 2.2 we present the classical CRA algorithm for the determinant and its modifications by [1] and [9].

2.1 Rational reconstruction and its application

A modular image of a rational number $\frac{a}{b} \bmod M$ can be computed by taking the modular images of a and b and applying the modular division. This fact can be written as

$$\frac{a}{b} = u \bmod M \Leftrightarrow a = bu \bmod M.$$

It should be noticed is that the opposite procedure can also be performed. One can reconstruct the fraction $\frac{a}{b}$ where $\gcd(a, b) = 1, b > 0$ from it modular image u. The solution is usually not unique but when we additionally require that $|a| < \frac{N}{2}$, $b \leq \frac{M}{N}$, then there exists at most one solution, see [12, Ch.5].

The solution to the rational reconstruction problem can be computed by applying the extended Euclidean algorithm EEA which searches for the gcd of M and u. The procedure Ratrec(a,b,u,M, N, D) takes as the input modulus M, $u \in \mathbb{Z}$ and the bounds N and D, and returns a fraction $\frac{a}{b} = u \bmod M$ such that $|a| < N, b < D$ or FAIL if no such solution exists. The worst case complexity of *Ratrec* is thus the same as for the EEA algorithm i.e. $O\left(\log^2(M)\right)$ for the classical algorithm and $O(\log(M)\log(\log(M)))$ for the fast Euclidean algorithm, see [12, Ch.11]. We will use the notation EEA(M) for the complexity of the Extended Euclidean Algorithm with entries bounded by M.

In many application, the cost of rational reconstruction is usually small compared with the cost of computing u and M. The general scheme is to recursively compute u_k, M_k, where $M_k = p_1 p_2 \cdots p_k$ or $M_k = p^k$ until $M_k > 2ND$ and then to apply the rational reconstruction. The complexity of the procedure depends on the number k of steps, which can be quite large Reducing the number of steps can be the easiest way to enhance the performance of the

algorithm.

This can be seen on the example of the Dixon algorithm [2] to solve a linear system $Ax = b$ of integer equations. Let N, D be the bound for the numerator and denominator of x. In the classical approach we compute the p-adic approximation in $k > \log(N) + \log(D) + 1$ steps and then reconstruct the result, which gives the complexity $O\left(m^3(\log(m) + \log(\|A\|))^2\right)$ when we use the bound of Hadamard for D, N and assume $b \in O(1)$. See [15] for a detailed complexity study. In fact, the number of entries in x which we need to reconstruct can often be reduced, see [7].

One should however notice, that the bounds N and D can be much bigger than the actual result. The idea is therefore to apply the rational reconstruction periodically and check the solution for correctness. If $M_k = p^k$ is the modulus in the current step, the method of Wang [18] prompts us to set $\sqrt{\frac{M_k}{2}}$ as the current bound for numerators and the denominator in *Ratrec*. The algorithm is guaranteed to return the result if $M_k > 2\max(N(x)^2, D(x)^2)$, where $N(x), D(x)$ are the values of the numerator and the denominator. In the opposite case, $Ratrec(a, b, u, M_k, \sqrt{\frac{M_k}{2}}, \sqrt{\frac{M_k}{2}})$ should fail with large probability. If we apply Wang's idea to the p-adic lifting we can reduce the number of steps to $k = 2\log_p(\max(N(x), D(x))) + 1$ and the complexity becomes $O\left(m^\omega + m^2 k \log(m\|A\|)\right)$ Current work on this field focus on further reducing the number of steps in the case when $N(x) \ll D(x)$ or $D(x) \ll N(x)$. A purely heuristic idea is to use the bounds $\sqrt{\frac{M_k}{2}\frac{N}{D}}$, $\sqrt{\frac{M_k}{2}\frac{D}{N}}$ instead of $\sqrt{\frac{M_k}{2}}$. For other approaches, see [14, 17].

2.2 Integer Determinant Algorithms

For an integer matrix A one has several alternatives to compute the determinant. The classical approach is to use Chinese Remaindering Algorithm (CRA) to reconstruct the value from sufficiently many modular images. The modular determinant is computed by LU factorization in the time $O(n^\omega)$, where n is the matrix dimension.Each step of the algorithm consist of computation mod p_i and a reconstruction of the determinant mod $p_1 \cdots p_i$ by the Chinese Remaindering Theorem. The computation is stopped when the early termination (ET) condition is fulfilled i.e. the reconstructed result rests the same for several iterations. The algorithm is Monte Carlo type, where the probability of success is controlled by the number of repetitions. See [4, 9] for a detailed description.

A mixture of CRA loop and Dixon p-adic lifting is used to compute the integer determinant in [1] and in the hybrid algorithm of [9]. The principle is to reduce the value reconstructed by CRA algorithm by computing a large fraction of the determinant. By solving several linear systems we can compute some largest invariant factors $s_m, \ldots s_{m-i}$. Their product π is potentially a large part of the determinant. An early terminated CRA loop which reconstructs $\det(A)/\pi \bmod p_0 p_1 \cdots p_i$ usually requires only a few modular determinant computations. Informally, the algorithm can be described as follows.

1. For $i = 0$ to k do
   (a) Solve $Ax_i = b_i$ by Dixon p-adic lifting to find $s_m, \ldots s_{m-i}$;
   (b) $\pi = s_m \cdots s_{m-i}$;
   (c) Run CRA for several iterations to determine $\det(A)/\pi$;
   (d) if ET break;
2. Run another determinant algorithm to get the result;

Here, k should not exceed the expected number of invariant factors which is $O(\sqrt{\log(n)})$ see [9]. The expected complexity of the hybrid determinant algorithm [9] for random dense matrices is $O\left(n^3 \log^{2.5}(n\|A\|)\right)$. In the worst case (step 2) we can choose between the CRA algorithm and the algorithms of [16, 10, 13]. In fact, in the expected case we do not need to run this step. The experiment proved that thanks the adaptive solutions this algorithm performs better than other implementation for a larger group of matrices.

3. RATIONAL DETERMINANT ALGORITHMS

The algorithms to compute the rational determinant are based on the ideas described in section 2. We present four main strategies to compute the rational determinant. They all use CRA which allows us to compute the determinant of the matrix modulo a product $p_1 \cdots p_k$ of primes. Then the first variant uses the rational reconstruction to obtain the rational result. In order to make use of Early Termination condition we have to precondition the determinant to obtain its integer multiplication. Preconditioning of the matrix allows us to use the integers determinant algorithms. The application of two determinant algorithms is studied

here. The common requirements for all algorithm are shown in 3.0. The algorithms are Monte Carlo type due to the early termination used.

Requirements 3.0

Require: A - an $m \times m$ rational matrix;
Require: D_i, $i = 1 \dots m$ - the common denominator of the entries of the ith row (column);
Require: N, D - the bounds for the numerator and the denominator of $\det(A)$, $D = \pi D_i$;
Require: A set P of random primes;
Require: $Ratrec(a, b, u, M, N, D)$ - a procedure which reconstructs $\frac{a}{b} = u \bmod M, a < N, b < D$ or returns FAIL.
Ensure: $\det(A)$ - the determinant of the matrix.

The effectiveness of our methods depends heavily on the number of modular determinants computed and thus on the bound N and D for the numerator and the denominator of the determinant. One can compute D as the product of lcm of all denominators in a row (or a column). Then N can be computed as $D \cdot H$, where H is the Hadamard bound for matrix A. One should notice that the bounds can be largely overestimated. Thus, we proposed output-dependant approach which allows us to reduce the number of iteration.

The first idea is to employ the CRA scheme and compute the determinant for the modular images of a rational matrix. In the case when the determinant is rational, early termination condition never holds. Instead, we have to compute the bounds D and N for the denominator and numerator of the determinant. As soon as the product of primes $M = p_1 \cdots p_k$ overcomes $2ND$ we can apply rational reconstruction and reconstruct the determinant from the modular image. We can also use an output dependent rational reconstruction as described in section 2.1. This strategy is presented as algorithm RatLU. An early termination in the rational case would required applying the rational reconstruction from time to time with the bounds $N = D = \sqrt{\frac{M_k}{2}}$ and wait for the result to re-occur. This leads to solution when $M > 2\max\{n^2, d^2\}$, where n, d are the numerator and denominator of the determinant.

The second method can use the denominator bound D to make the CRA loop look for an integer value. Again, we compute the modular image of a rational matrix A but this time we call CRA to look for $D \times \det(A)$ which is integer. Now the classic ET condition can be used and the result is obtained as soon as $M > n\frac{D}{d}$. The effectiveness of this method

Algorithm 3.1 RatLU

$i = 0, k = 0, n = 0, d = 1, M = 1, u = 0$;
repeat
  $++i$; Get p_i from P;
  Compute $A_i = A \bmod p_i$;
  Compute $u_i = \det(A_i)$;
  Reconstruct $u = \det(A) \bmod Mp_i$ using M, u, u_i, p_i, $M = Mp_i$;
  if $i = k^2$ **then**
    s = Ratrec$(n, d, u, M, \sqrt{\frac{M}{2}}, \sqrt{\frac{M}{2}})$;$++k$;
    if s $\neq$ FAIL **then** return n, d; **end if**
  end if
until $M > 2ND$
status = Ratrec(n, d, u, M, N, D);
if status $\neq$ FAIL **then** return n, d; **end if**

depends therefore on the exactness of denominator bound D. Experimental results show that it is sufficient in practice, see sec. 5 table 2. This strategy is presented as algorithm PrecDetLU.

Algorithm 3.2 PrecDetLU

$i = 0; M = 1; u = 0$;
repeat
  $++i$;Get p_i from P;
  Compute $A_i = A \bmod p_i$;
  Compute $u_i = D \cdot \det(A_i)$;
  reconstruct $u = D \cdot \det(A) \bmod Mp_i$ using M, u, u_i, p_i, $M = M \cdot p_i$
  if ET holds **then** return $\frac{u}{\gcd(u,D)}, \frac{D}{\gcd(u,D)}$; **end if**
until $M > 2ND$
return $\frac{u}{\gcd(u,D)}, \frac{D}{\gcd(u,D)}$;

The last two strategies require an integer matrix $\tilde{A}$ which can be obtained by preconditioning the rational matrix A. In order to obtain an integer matrix, the easiest way would be to take matrix $A' = D(A)A$, where $D(A)$ is the common denominator of all entries. In the general case, where the entries of A are fractions $\frac{a_{ij}}{b_{ij}}$ with numerator and denominator bounded by $\|A\|$, this is not the best choice as the size of $D(A)$ can be as large as $O(\|A\|^{m^2})$. This causes $\log(\|A'\|)$ to be $O(m^2)$. Moreover, the denominator approximation is $D(A)^m$ in this case, which is $O(m^3)$ in size. We have already defined a tighter bound for the denominator of $\det(A)$ by πD_i, which is $O(m^2)$ in size. Now, if we want to use the integer matrix $\tilde{A}$ then we can precondition A by taking $\tilde{A} = A\,\mathrm{diag}(D_i)$, where D_i are the common denominators of the rows (or $\tilde{A} = \mathrm{diag}(D_i)A$, where D_i are the common denomina-

tors of the columns). For the preconditioned matrix $\tilde{A}$ all integer determinant algorithms can be applied. In particular the hybrid determinant algorithm of [9] can be used. The drawback of this approach is the size of the coefficients of $\tilde{A}$ compared to A, see section 5 table 1. This forced us to use early terminated rational reconstruction for system solving in the Dixon p-adic lifting algorithm. The strategies that use the CRA loop or the hybrid algorithm are presented as algorithms PrecMatLU and PrecMatDixon respectively.

Algorithm 3.3 PrecMatLU

$i = 0; M = 1; u = 0;$
Compute $A = A\,\mathrm{diag}(D_i)$ (or $\mathrm{diag}(D_i)A$)
repeat
  Get p_i from P;
  Compute $A_i = A \bmod p_i$;
  Compute $u_i = \det(A_i)$;
  reconstruct $u = \det(A) \bmod Mp_i$ using M, u, u_i, p_i, $M = M \cdot p_i$
  if ET holds **then** return $\frac{u}{\gcd(u,D)}, \frac{D}{\gcd(u,D)}$; **end if**
until $M > 2ND$
return $\frac{u}{\gcd(u,D)}, \frac{D}{\gcd(u,D)}$;

Algorithm 3.4 PrecMatDixon

Compute $A = A\,\mathrm{diag}(D_i)$ (or $\mathrm{diag}(D_i)A$);
Compute $u = \det(A)$ by HybridDet [9];
return $\frac{u}{\gcd(u,D)}, \frac{D}{\gcd(u,D)}$;

4. COMPLEXITY ANALYSIS

In this section we study the complexity of the algorithms presented in section 3. In subsection 4.1 we present the analysis of the general case, where we assume that the entries of the matrix are fractions with numerators and denominators bounded by $\|A\|$. Then, in subsection 4.2, we will focus on two special cases i.e. matrices of decimal fractions and Hilbert matrices.

The complexity of the strategies described in section 3 depends on the number of iterations required by the **while** loop of CRA. Then, depending on the strategy, we have to include the cost of computing the homomorphic image of the matrix, the cost of the rational reconstruction or the cost of p-adic lifting. If we use the early termination condition, the number of steps required for the computation of $\det(A)$ depends on the values: m - the size of the matrix, n, d - the real values of the numerator and denominator of $\det(A)$ and D - the bound for the denominator. The cost of homomorphic imaging depends on the maximum norm of the matrix i.e. $\|A\| = \max\{\|a_{ij}\|, b_{ij}\}$ and $\|\tilde{A}\|$.

4.1 General case

We start this section by the analysis of the rational homomorphic imaging schemes. We have the following lemma.

LEMMA 4.1. *Let p be a word-size prime. Then the complexity of computing the modular image at p for a rational matrix A is $O(m^2(\log(\|A\|))+\mathrm{EEA}(p))$ word operations.*

PROOF. For a matrix without a pattern we compute an image for all m^2 entries. For a rational fraction the cost is $O(\log(\|A\|))$ for the computation of the modular image of the numerator and denominator and $\mathrm{EEA}(p) = O(\log(p)\log(\log(p)))$ for the modular inverse computation by fast extended Euclidean algorithm. Therefore for a word-size $\|A\|$ the cost of computing the image is $O(1)$ yet important, due to the constant for computing the inverse of an element mod p. □

For the integer case, the cost is $\log(\|\tilde{A}\|))$. We can notice that $\log(\|\tilde{A}\|)$ can be $O(m\log(\|A\|))$ in the worst case, so the complexity of homomorphic imaging in terms of m is $O(m^2)$ for the rational and $O(m^3)$ in the integer case. But if $\|\tilde{A}\| < p$ the cost of imaging for one element is 1. Thus, if both $\|\tilde{A}\|$ and $\|A\|$ are less than p, the complexity of the homomorphic imaging becomes $m^2\,\mathrm{EEA}(p)$ for the rational and m^2 for the integer case. In this case, it is better to use integer imaging. On the other hand, if matrix A is structured, for example it is Hankel-type, we have the complexity $m\,\mathrm{EEA}(p)$ for rational imaging. Due to the preconditioning, we loose the structure pattern for $\tilde{A}$ and the complexity of integer imaging rests without change. Finally we notice, that for sparse matrices with Ω elements, we can take Ω instead of m^2 in the complexity formula.

Putting it together we have the following theorem.

THEOREM 4.2. *The worst case complexities of the strategies for computing the determinant of a rational matrix A of size m are*

1. $O\left(k(m^2\log(\|A\|) + m^\omega)\right) + O^\sim\left(k\sqrt{k}\right)$ *for RatLU, where $O^\sim$ hides some $\log(k)$ factors;*

2. $O\left(\log(\frac{D}{d}n)(m^2\log(\|A\|)+m^\omega)\right)$ *for PrecDetLU;*

3. $O\left(\log(\frac{D}{d}n)(m^2\log(\|\tilde{A}\|)+m^\omega)\right)$ *for PrecMatLU;*

4. $O^\sim(x(m^2(\log(m)+\log(\|\tilde{A}\|))+mx^{\frac{1}{2}})+O(\log(\frac{D}{d}\frac{n}{s_m}+1)(m^2\log(\|\tilde{A}\|)+m^\omega))$ *for PrecMatDixon, where* $s_m = s_m(\tilde{A})$ *and* $x \in m(\log(m\|\tilde{A}\|\|b\|)$ *is the size of solution to* $\tilde{A}x = b$.

Here $\tilde{A}$ *is equal to* $A\,\mathrm{diag}(D_i)$ *as in section 3;* n, d *are the numerator and denominator of* $\det(A)$ *and* $k = O(\max(\log(n), \log(d)))$.

PROOF. The complexities can be obtained by a careful examination of the number of CRA steps. The result for alg. RatLU takes into account the cost of the rational reconstruction which is performed at most $O(\sqrt{k})$ times. In alg. PrecMatDixon we introduce x to estimate the cost of early terminated p-adic lifting. The size of x can generally vary depending on the choice of b but is $O(m\log(m\|\tilde{A}\|\|b\|))$ in the worst case. To further evaluate the worst case complexity of alg. PrecMatDixon we assumed that *HybridDet* continues to use CRA loop in the worst case. Thus the number of iterations $O(\log(\frac{D}{d}\frac{n}{s_m}))$ and the complexity. □

Special care should be taken if we consider the use of alg. PrecMatDixon. As $\|\tilde{A}\|$ can potentially be $O^\sim(m)$ in size and with a pessimistic bound on x, its worst case complexity can be $O^\sim(\log(m^4))$, which is worse than for the CRA computation. Nevertheless, the gain of computing s_m can be important, as it is the case in the *HybridDet* algorithm, see [9].

4.2 Complexity in the special cases

By the precedent remarks it should be visible, that the analysis of the strategies should be divided into two main cases. One would consist of the matrices, whose entries are given by decimal fraction, or more generally, where the common denominator of all entries, the common denominator of the rows and the norm of A are of the same order i.e. $D(A) = O(D_i) = O(\|A\|)$. In the other case matrix entries are given as fractions with different denominators. We will study the complexity of the algorithms on the example of Hilbert matrices.

In the case of matrices of decimal fractions let us further assume that $\|A\|$ is $O(1)$. This would be the case of numerous ill-conditioned matrices emerging from different applications in science and engineering. In order to better describe the differences between the algorithms, we include the cost of EEA when it is relevant. The theorem is a straightforward consequence of theorem 4.2.

THEOREM 4.3. *The complexities of the strategies in the case when* $\|A\| = O(\|\tilde{A}\|) = O(1)$ *are:*

1. $O^\sim\left(k(m^2\,\mathrm{EEA}(p)+m^\omega+k\sqrt{k}\right)$ *for alg. RatLU;*

2. $O^\sim\left(\log(\frac{D}{d}n)(m^2\,\mathrm{EEA}(p)+m^\omega)\right)$ *for alg. PrecDetLU;*

3. $O^\sim\left(\log(\frac{D}{d}n)(m^2+m^\omega)\right)$ *for alg. PrecMatLU;*

4. $O^\sim\left(x(m^2\log(m)+mx^{\frac{1}{2}}))+\log(\frac{D}{d}\frac{n}{s_m})(m^2+m^\omega)\right)$ *for alg. PrecMatDixon.*

where k, x *are as in theorem 4.2.*

The analysis suggests that the algorithm PrecMatLU should be better than PrecDetLU (see 4.1 for the homomorphic image complexity). The performance analysis in section 5 confirms this observation. Furthermore, as long as the Smith form of $\tilde{A}$ is simple, we encourage the use of strategy PrecMatDixon. In particular, we can establish an equivalence between matrices A of random decimal fractions with e decimal places taken randomly an uniformly from the interval $[0,1]$ and matrices $\tilde{A}$, $\|\tilde{A}\| < 10^e$. This allows us to use the expected complexity of the hybrid algorithms of [9] as the expected complexity of the rational determinant computation by alg. PrecMatDixon. Also, the preconditioning should be used instead of strategy RatLU. For more details see section 5.

The other group consists of matrices with rational entries given by fractions with very different denominators. As a model case we can consider Hilbert matrices. Hilbert matrices are the matrices of the form $H_m = [\frac{1}{i+j-1}]_{i,j=1..m}$. They are benchmarks examples for many numerical methods. The formula for the determinant of a Hilbert matrix is well known and is given by the equation

$$\frac{1}{\det(H_m)} = \Pi_{k=1}^{m-1}(2k+1)\binom{2k}{k}^2.$$

THEOREM 4.4. *The complexities for rational determinant strategies in the case of Hilbert matrices are*

1. $O\left(m^2\log(m)(m^\omega + m\sqrt{\log(m)})\right)$ *for alg. RatLU;*
2. $O\left(m^{\omega+2}\log(m)\right)$ *for alg. PrecDetLU;*
3. $O\left(m^5)\log(m)\right)$ *for alg. PrecMatLU;*
4. $O(s_m m^3\log^2(m) + m^5\log(m))$ *for alg. PrecMatDixon.*

PROOF. One should notice that $\log(\frac{1}{\det(H_m)})$ is $O(m^2\log(m))$. The size of entries of H_m is $\log(\|H_m\|) = O(\log(m))$ and $\log(\|\tilde{H_m}\|) = O(m\log(m))$. $\square$

In the case of Hilbert matrices algorithm PrecDetLU - has the best time complexity and also performed best in the experiments, see section 5. Since the numerator is equal to 1, we only have to recover the size of the over-approximation. Experimental results show, that its size is equal to about 8% of the denominator size. Therefore, alg. PrecDetLU, PrecMatLU perform about 25 times less iterations than RatLU. As for the algorithm PrecMatDixon, the study of the Smith form of $\tilde{H_m}$ has revealed that it is quite complex, with about $2\sqrt{m}$ nontrivial factors and the size $\log(s_m(\tilde{H_m}))$ equal $O(m)$. Thus, it is not worth computing PrecMatDixon due to the high cost of the algorithm and poor gain.

5. PERFORMANCE COMPARISON

In this section we present the experimental results for four strategies from section 3. We have tested the performance of four strategies on three matrix sets: random, ill-conditioned and Hilbert matrices.

We generated the random matrices using Matlab procedure *rand*. The entries of the matrices are decimal fractions with 6 decimal places chosen randomly from the interval $[0, 1]$. The determinant of the resulting matrices is large in the absolute value. The result of the numerical procedure of Matlab is $\pm\infty$.

Ill-conditioned matrices have been chosen from the Matrix Market [20] Harwell-Boeing collection. We chose three sets: Grenoble, Astroph and Bcsstruc3. Grenoble set represents the results of the simulation of computer systems. The sizes of the matrices varies from 115 to 1107 and the condition numbers range from $1.5 \cdot 10^2$ in the case of the smallest matrix to $9.7 \cdot 10^7$ for the biggest. The decimal precision of the entries depends on the matrix and ranges from 1 to 5 decimal places. The determinants are close to 0. For these matrices, Matlab procedure *det* computes the result correctly up to the 5th decimal place. Since matrix entries seem to be represented as rounded expansions of rational numbers, we computed the determinant of the matrices "as is" and then we took continued fractions approximants of the entries with the same precision as the decimal fractions.

Astroph set describes the process of nonlinear radiative transfer and statistical equilibrium in astrophysics. The condition number is $3.6 \cdot 10^{17}$ for the small 180×180 matrix and $1.7 \cdot 10^{14}$ for the 765×765 one. The result of Matlab computation is $-\infty$. Bcsstruc3 gives dynamic analyses in structural engineering. All matrices are symmetric. The condition number is about 10^{11} for matrices 19 and 20 and 10^5 for matrix 22. The result of Matlab computation is ∞.

We split the analysis of the performance of the algorithms in three phases. First, we will consider the cost of rational-modular vs. integer-modular imaging and compare it with the results for $\|A\|$ and $\|\tilde{A}\|$. Then we will take a look on the numerator and denominator approximations D and N computed by our algorithms. Finally, we give the timings for all strategies and compare their performance.

As we can see in table 1, the time of computing an integer image can be several times shorter than for the rational image provided that the size of preconditioned matrix is still small. This is not the case for Hilbert matrices of dimension ≥ 250 , when the time of rational image computation is better. Furthermore, for structured matrices, like Hilbert, we can reduce the number of images computed. For a Hankel-type matrix, there are only $2n - 1$ images to compute, which makes the cost of imaging negligible.

The performance of the algorithms depends on the accuracy of denominator approximation used. For the bound $D = \pi D_i$, the resulting size of the over-approximation is shown in table 2, column 4. In algorithm PrecMatDixon we additionally approximate the numerator by computing $s_m(\tilde{A})$. In this case we are interested in the value $App(N) = s_m(\tilde{A})$ and $\frac{D}{d}\frac{n}{s_m(\tilde{A})}$ which we compute instead of the numerator. As we can see in the table, the quality of the approximation of the denominator depends on the matrix and ranges from 1-2% in the case of sparse matrices in the Grenoble set, to 80% for Bccstk matrices. For Hilbert matrices the approximation is quite efficient, the over-approximation is always less than 10%. Table 3 shows that despite

A	RatIm	IntIm	IntIm/RatIm	$\log(\|A\|)$	$\log(\|\tilde{A}\|)$
bccstk817	0.14587	0.03126	4.66696	60	66
bccstk485	0.05189	0.01123	4.61980	65	69
bccstk138	0.00280	0.00050	5.53681	42	42
mmca180	0.00808	0.00120	6.74795	77	76
mccf765	0.13222	0.03215	4.11219	70	68
grenoble115	0.00162	0.00019	8.51887	19	19
grenoble185	0.00746	0.00096	7.8125	19	19
grenoble216a	0.01055	0.00145	7.25	2	1
grenoble216b	0.0105	0.00106	9.90055	19	19
grenoble343	0.02641	0.00507	5.21053	2	1
grenoble512	0.0588	0.0126	4.66667	2	1
grenoble1107	0.26762	0.05682	4.70958	16	16
random200	0.037	0.003	11.692	19	19
random500	0.330	0.028	11.831	19	19
random800	0.599	0.071	8.436	19	19
random1000	0.934	0.111	8.452	19	19
hilbert100	0.00414	0.00255	1.62264	7	289
hilbert200	0.02174	0.01984	1.09552	8	567
hilbert250	0.03481	0.03629	0.95942	8	714
hilbert300	0.05093	0.05967	0.85350	9	847
hilbert400	0.09307	0.13343	0.69756	9	1134
hilbert600	0.21485	0.41759	0.51450	10	1711
hilbert800	0.38839	0.94920	0.40917	10	2294
hilbert1000	0.61425	1.81285	0.33883	10	2866

Table 1: Comparison of the times (in seconds) for homomorphic imaging are given in columns RatIm (for rational) and IntIm (for integer). The ratio of the timings is given in column 3. Last two columns give the size of entries for A and $\tilde{A}$. Matrix size is included in its name.

the size of the over-approximation, preconditioning allow us to gain enough to beat the naive RatLU algorithm. If the size of $\|\tilde{A}\|$ is small, as is the case for sparse matrices, we can compute $s_m(\tilde{A})$ at a relatively low cost and efficiently approximate the numerator.

The timings for all algorithms are shown in table 3. The results for Hilbert matrices agree with the complexity estimation in Thm. 4.4. Note that alg. PrecMatDixon is usually the best for the matrices from MatrixMarket collection.

For the Grenoble set, the approximation by continued fractions allowed quite well, in our opinion, to reconstruct the orginal rational matrix connected to the problem. Despite the difference in properties, the running times for the decimal and continued fractions variants were simmilar. However, although the matrices were close in the maximum norm, the determinants ratio reached as much as 2 in the case of grenoble1107.

In figure 5 we present the results of the determinant computation for Hilbert matrices. We compare the timings for algorithm RatLU, PrecDetLU, PrecMatLU, PrecMatDixon, and the Maple LinearAlgebra::Determinant algorithm with *method=rational*. The best performance is observed for a variant of algorithm PrecDetLU which takes into account the

A	$\log(d)$	$\log(n)$	$\log(D/d)$	$\frac{\log(D/d)}{d}$	$\log(App(n))$	$\log(\frac{Dn}{dApp(N)})$
bccstk817	7845	36169	6294	0.802	25923	16540
bccstk485	3903	21921	2538	0.650	16225	8234
bccstk138	2576	5040	139	0.054	3880	299
mmca180	1663	7341	571	0.343	7375	537
mccf765	5503	32451	2626	0.477	32483	2594
grenoble115	2243	2136	36	0.016	1526	646
grenoble185	3072	2785	3	0.001	2777	11
grenoble216a	423	131	9	0.021	124	16
grenoble216b	4110	3278	193	0.047	683	2788
grenoble343	678	209	8	0.012	201	16
grenoble512	1009	303	15	0.015	306	12
grenoble1107	15639	14002	2707	0.173	7184	9525
random200	3986	4255	0	0	4255	0
random500	9961	10952	4	0	10956	0
random800	15944	17797	1	0	17798	0
random1000	19931	22407	0	0	22404	3
hilbert100	19737	1	1690	0.086	130	1561
hilbert200	79472	1	6493	0.082	290	6204
hilbert300	179207	1	14323	0.080	424	13900
hilbert400	318942	1	26509	0.083	563	25947
hilbert600	718412	1	59948	0.083	848	59101
hilbert800	1277881	1	103581	0.081	1133	102449
hilbert1000	1997351	1	164550	0.082	1424	163127

Table 2: The size of the numerator n and denominator d of $\det(A)$, the size of the denominator over-approximation D/d computed by PrecDetLU and PrecMatLU; the numerator approximation $App(n)$ obtained as s_m in PrecMatDixon, and the size of the part remaining to compute. s_m depends on n and the over-approximation D/d.

Matrix	RatLU	PrecDetLU	PrecMatLU	PrecMatDixon
bccstk817	*	789.02	553.624	**318.62**
bccstk485	278.964	143.888	95.836	**57.144**
bccstk138	4.12	1.868	1.324	**0.764**
mmca180	14.404	5.896	3.644	**1.604**
mccf765	*	585.724	416.352	**128.24**
grenoble115	1.444	0.591813	0.456	**0.288**
grenoble185	5.86	2.34	1.456	**0.468**
grenoble216a	1.052	0.268	**0.248**	0.26
grenoble216b	10.448	3.852	**2.204**	2.128
grenoble343	4.292	0.924	0.832	**0.732**
grenoble512	14.844	2.868	2.48	**1.072**
grenoble1107	*	698.436	519.368	**367.448**
random200	24.096	10.776	3.996	**2.980**
random500	432.448	180.448	71.492	**54.996**
random800	1715.316	789.154	331.008	**205.188**
random1000	*	1572.024	662.956	**403.232**
hilbert100	17.860	0.664	**0.548**	0.712
hilbert200	330.280	11.104	**10.52**	11.312
hilbert300	*	**59.144**	65.236	66.872
hilbert400	*	**200.844**	252.676	265.276
hilbert600	*	**1072.754**	1664.738	1735.574
hilbert800	*	**3476.188**	6299.98	8830.372
hilbert1000	*	**8870.534**	18466.348	19328.66

Table 3: Timing comparison for 4 rational determinant strategies. All times in seconds. Best times in bold.

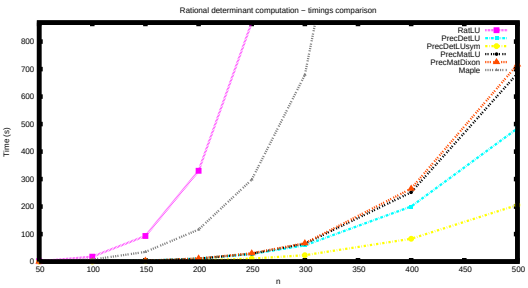

Figure 1: Comparison of the timings for the exact computation of the rational determinant of Hilbert matrices. The results for algorithms RatLU, PrecDetLU, PrecMatLU and PrecMatDixon implemented in LinBox and Maple Determinant procedure are shown. Algorithm PrecDetLU is used in the classic and symmetric variant, which takes into account the Hankel structure of the matrix. All times in seconds.

Hankel structure of the matrix.

6. CONCLUSIONS

It this paper we have presented four strategies for exact computation of the determinant of a rational matrix. We have evaluated the performance of these algorithms on several sets of matrices. The performance of the algorithms suggests that there exists a clear division between the matrices given as a rational approximation (by decimal fractions) of real valued matrices and the matrices with a great diversity of the denominators of the entries. For the first case, matrix preconditioning which leads to a integer matrix is proposed, which allows us to use integer determinant algorithms, see solution PrecMatDixon. For the second case, determinant preconditioning is preferred, which does not lead to matrix coefficient blow-up. In general, preconditioning proved more useful than rational reconstruction tools, although better early termination methods where the modulus M is linear in the size of the output n and d can bring a change, see [14, 17].

An adaptive solution should be able to choose the best storage method and homomorphic imaging scheme, and work independently of the determinant over-approximation.

We propose the following solution, which incorporates the elements of all algorithms

1. Compute $D = \pi D_i$, $\tilde{A}$; set $N = 1$;
2. If $\log_p(\|\tilde{A}\| < C)$ compute $N = s_m(\|\tilde{A}\|)$ - see alg. PrecMatDixon
3. Compute the modular image of the rational matrix A and integer matrix $\tilde{A}$, determine whether to use PrecDetLU or PrecMatLU based on the timings.
4. Run the ET CRA loop for $\frac{D}{N} \cdot \det(A)$ using PrecDetLU or PrecMatLU.
5. From time to time check by rational reconstruction the early termination condition on $\det(A)$ - see RatLU.

This algorithm can be further developed to compute other invariant factors as in alg. PrecMatDixon if relevant. Notice, that the cost of introducing solution RatLU to the adaptive algorithm is virtually that of rational reconstruction.

Further work can include intertwining algorithms RatLU and PrecDetLU to include the use of less exact determinant preconditioners, which potentially are not a multiple of d. The aim would to reduce a factor of the denominator by preconditioning and reconstruct the remaining part by rational reconstruction. The strategy should be effective, if the over-approximation caused by preconditioning is reduced but a large fraction of the denominator is obtained at the same time. For example, $D = \pi D_i / gcd(D_i)$ could be considered.

Further work can then focus on the implementation of the solution in the case of sparse matrices and on the parallelization of the algorithms.

In this paper we have considered the case of dense matrices in the analysis of the complexity of the strategies as well as in the implementation. However, sparse matrix counterparts of the algorithms can also be used. For the modular determinant computation one could used the algorithm of Wiedemann [19] that computes the determinant by finding the characteristic polynomial of the matrix. In alg. PrecMatDixon the sparse solver of [11] can be used.

The strategies described in this paper contain elements that allow parallelization. This concerns in particular the CRA loop, where several iterations can be performed at the same time, see [3]. The question of an optimally distributed early termination in the case of integer Chinese reconstruction

(alg. PrecDetLU, PrecMatLU, PrecMatDixon) as well as the rational reconstruction (alg. RatLU) has not yet been addressed. For a parallel p-adic lifting for alg. PrecMatDixon, see [5].

In this paper we have developed and compared four strategies to compute the rational determinant of a matrix. We have proposed two preconditioning methods that allow us to transfer the problem from rational to integer domain. We believe that the approach described in this article can also be applied in other problems of exact computation in rational numbers such as rank computation or system solving.

7. REFERENCES

[1] J. Abbott, M. Bronstein, T. Mulders. Fast deterministic computation of determinants of dense matrices. *ISAAC'1999*, pp. 197-204, ACM Press, 1999.

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

[3] J.G. Dumas. Calcul parallele du polynome minimal entier en Athapascan-1 et Linbox. *RenPar'2000*. pp119-124. 2000.

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

[5] J.G. Dumas, W. Turner, Z. Wan. Exact Solution to Large Sparse Integer Linear Systems. *ECCAD'2002*, 2002.

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

[7] 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*. 2002.

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

[9] J.G. Dumas, A. Urbańska. An introspective algorithm for the integer determinant. Research report. http://arxiv.org/abs/cs.SC/0511066.

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

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

[12] J. von Gathen, J. Gerhard. Modern Computer Algebra. *Cambridge University Press* 1999.

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

[14] S. Khodadad and M. Monagan. Fast rational function reconstruction. *ISSAC'2005*, pp. 184–90. ACM Press, New York, 2006.

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

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

[17] Z. Olesh, A. Storjohann. The vector rational function reconstruction problem. *WWCA 2006*.

[18] P.S. Wang. A p-adic Algorithm for Univariate Partial Fractions. *Proc. of the 4th ACM Symp. on Symb. and Alg. Comp.* pp 212-217. 1981.

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

[20] Matrix Market. http://math.nist.gov/MatrixMarket/

[21] GNU Multiprecision Package. http://www.swox.com/gmp/

[22] Givaro library. http://ljk.imag.fr/CASYS/LOGICIELS/givaro/