Exact Determinant of Integer Matrices

Takeshi Ogita
Department of Mathematical Sciences, Tokyo Woman's Christian University
Tokyo 167-8585, Japan, ogita@lab.twcu.ac.jp

Abstract. This paper is concerned with the numerical computation of the determinant of matrices. Algorithms for computing enclosures and the exact values of the determinant of integer matrices are proposed. The algorithms can treat extremely ill-conditioned matrices. To achieve it, an accurate algorithm for inverse LU factorization is used. Then accurate and verified results of the determinant can efficiently be obtained for much wider range of the problems. A possibility for verifying the singularity of integer matrices is also discussed. Numerical results are also presented showing the performance of the proposed algorithms.

Keywords: Determinant, verified numerical computation, verification of singularity

1. Introduction

Computing the determinant of a given matrix has several applications. For example, it is used for computational geometry (cf. e.g. (Brönnimann et al., 2001)), in which it is essential to check the sign of the determinant. Moreover, it is also used for Chebyshev systems (Smith, 1969), robust control methods (cf. e.g. (Smagina and Brewer, 2002)) and nuclear lattice simulations (cf. e.g. (Lee and Ipsen, 2003)).

In this paper, we restrict A to be an integer matrix, i.e. $A \in \mathbb{Z}^{n\times n}$. Then the determinant of A is also an integer. Such a case has been discussed in (Pan and Yu, 2001; Kaltofen and Villard, 2004; Emiris and Pan, 2005) and the literatures cited there. Of course, we can efficiently obtain an approximation $\widetilde{d}$ of $\det(A)$ via LU factorization of A by floating-point arithmetic. Although the quality of such an approximation is usually satisfactory, it sometimes becomes very poor, which basically depends on the condition number of A. The problem is that we cannot know how accurate it is from the result itself.

The aim of the paper is to obtain the *exact* value $d = \det(A) \in \mathbb{Z}$. For this purpose, we develop a verified numerical algorithm for calculating a rigorous enclosure $[\underline{d}, \overline{d}]$ such that

$$\underline{d} \le \det(A) \le \overline{d} \quad \text{with } \lceil \underline{d} \rceil = \lfloor \overline{d} \rfloor = d, \tag{1}$$

where $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ denote the floor and ceiling functions, respectively. Then it implies $\det(A) = d$. It is much more difficult than to simply compute $\widetilde{d}$, since we need to obtain a verified and accurate result of the determinant.

Let $\mathbf{u}$ be the unit roundoff for floating-point arithmetic. In IEEE standard 754 double precision, $\mathbf{u} = 2^{-53}$. Let $\kappa(A)$ be the condition number of A defined by $\kappa(A) := \|A\| \cdot \|A^{-1}\|$. In this paper, we propose an algorithm for computing an accurate and verified enclosure of $\det(A)$ with A

4th International Workshop on Reliable Engineering Computing (REC 2010)
Edited by Michael Beer, Rafi L. Muhanna and Robert L. Mullen

ISBN: 978-981-08-5118-7. *Published by* Research Publishing Services.
doi:10.3850/978-981-08-5118-7_038

being extremely ill-conditioned, i.e. $\kappa(A) \gg \mathbf{u}^{-1}$. The proposed algorithm uses an accurate matrix factorization presented in (Ogita, 2009), which is based on standard algorithms for numerical linear algebra and accurate dot product. Thus the algorithm can benefit from optimized numerical libraries such as BLAS and LAPACK.

We also discuss a possibility for verifying the singularity of integer matrices. To prove the singularity of a matrix is known to be very difficult in numerical computations since it is an ill-posed problem, i.e. arbitrary perturbation for a singular matrix makes it nonsingular in general.

Finally, we present numerical results showing the performance of the proposed algorithms with some comparisons to existent algorithms.

2. Verification theory

Some verified numerical methods for calculating the matrix determinant have been proposed, e.g. in (Brönnimann et al., 2001; Rump, 2005). We first explain their methods. Both methods are based on LU factorization. Let A be a real $n \times n$ matrix. Let L, U and P be computed LU factors such that

$$PA \approx LU,$$

where L is a unit lower triangular matrix, U is an upper triangular matrix, and P is a permutation matrix following partial pivoting. Let further X_L and X_U be approximate inverses of L and U, respectively.

In (Brönnimann et al., 2001), the following lemma is presented:

LEMMA 1. *Let G be a real square matrix with $\|G\| < 1$ for some fixed matrix norm. Then $\|(I + G)^{-1}\| \le 1/(1 - \|G\|)$ and $\det(I + G) > 0$.*

Put $R := X_U X_L P$, which can be regarded as an approximate inverse of A. Suppose $\|RA - I\| < 1$. Then $\det(R) \neq 0$ and

$$\det(A) = \frac{\det(RA)}{\det(R)} = \frac{\det(RA)}{\det(X_U) \cdot \det(X_L) \cdot \det(P)} = \det(P)\frac{\det(RA)}{\det(X_U)}. \tag{2}$$

Note that $\det(X_L) = 1$ and $|\det(P)| = 1$. Using Lemma 1 yields $\det(RA) > 0$. Thus the sign of $\det(A)$ can be predicted as

$$\mathrm{sign}(\det(A)) = \det(P) \cdot \mathrm{sign}(\det(RA)) \cdot \mathrm{sign}(\det(X_U)).$$

Similarly to (2), Rump has presented an efficient verification method (Rump, 2005), which is based on the following fact: Let $B := X_L P A X_U$. Then

$$\det(B) = \det(P) \cdot \det(A) \cdot \det(X_U).$$

If $\det(X_U) \neq 0$, then

$$\det(A) = \det(P)\frac{\det(B)}{\det(X_U)}. \tag{3}$$

In (2) and (3), it can be expected that RA and B are nearly the identity matrix I, which is almost diagonal, and $\det(RA) \approx 1$ and $\det(B) \approx 1$. So, Gershgorin's circle theorem gives (enclosures of) the product of all eigenvalues of RA and B, which implies enclosures of $\det(RA)$ and $\det(B)$. It turns out that an enclosure of $\det(A)$ can also be computed using (2) or (3).

Based on (2), we present an algorithm for verifying the determinant of A. The program is written in INTLAB (Rump, 1999), a Matlab toolbox for interval computations.

ALGORITHM 1. *Enclosure of* $\det(A)$ *based on (2).*

```
function d = vdet1(A)
  [L,U,p] = lu(A,'vector'); % LU factorization
  I = speye(size(A));       % Identity matrix
  XL = I / L;               % Left inverse of L
  XU = I / U;               % Left inverse of U
  R = intval(XU)*XL;        % Approximate inverse of A(p,:)
  B = R*A(p,:);             % Inclusion of XU*XL*A(p,:)
  c = mid(diag(B));
  r = mag(sum(B-diag(c),2));
  g = midrad(c,r);          % Gershgorin's circles
  u_prod = prod(intval(diag(XU)));
  d = det(I(p,:))*prod(g)/u_prod;
```

REMARK 1. *The last two lines in the algorithm can easily cause over/underflow. We can avoid it, for example, by representing* `d` *as* $\mathtt{d} = f \cdot 2^e$ *with* $0 \le |f| < 1$ *and* e *being a corresponding integer.*

The following is a slightly modified version of the Rump's algorithm described in (Rump, 2005, p. 214).

ALGORITHM 2. *Enclosure of* $\det(A)$ *based on (3).*

```
function d = vdet2(A)
  [L,U,p] = lu(A,'vector'); % LU factorization
  I = speye(size(A));       % Identity matrix
  XL = I / L;               % Left inverse of L
  XU = U \ I;               % Right inverse of U
  B = XL*intval(A(p,:))*XU; % Inclusion of XL*A(p,:)*XU
  c = mid(diag(B));
  r = mag(sum(B-diag(c),2));
  g = midrad(c,r);          % Gershgorin's circles
  u_prod = prod(intval(diag(XU)));
  d = det(I(p,:))*prod(g)/u_prod;
```

The point of Algorithms 1 and 2 is to use standard numerical algorithms such as LU factorization and solving linear systems as much as possible. So, the verification process in the algorithms relies on the fact that the quality of the results of the standard algorithms in pure floating-point arithmetic

is usually good. On the other hand, if an interval Gaussian elimination, in which all operations of Gaussian elimination are done in interval arithmetic instead of pure floating-point, is used, then the worst-case accumulation (frequently an exponential growth) of rounding errors occurs, so that it can only apply to small dimensional problems regardless of condition number of A. See (Rump, 2005) for details.

Basically, the above-mentioned approach can apply if $\kappa(A) < \mathbf{u}^{-1}$. However, if A is ill-conditioned, then L and U may not be good LU factors, i.e. X_L and X_U cannot work as good preconditioners for A. In such cases, we need some higher precision arithmetic. In the next section, we will discuss how to solve the ill-conditioned problems.

3. Ill-conditioned case

In this section, the case of $\kappa(A) \gg \mathbf{u}^{-1}$ is treated. In about 1984, Rump derived a method for inverting an extremely ill-conditioned matrix. Oishi et al. (Oishi et al., 2007) presented a part of proof why (modified) Rump's method works so well. Moreover, Rump (Rump, 2009) gave detailed analysis for the original Rump's method.

Although the Rump's method can compute an accurate approximate inverse of A with $\kappa(A) \gg \mathbf{u}^{-1}$, it is not suited for calculating the determinant of A. Because, it is not so easy to extract the information of the determinant from the inverse of A. On the other hand, with the same philosophy as the Rump's method, the author has recently proposed a new method for calculating an accurate inverse LU factors of an ill-conditioned matrix in (Ogita, 2009). For a given torelance $\varepsilon_{\rm tol}$, the method computes accurate LU factors X_L, X_U and P of A^{-1} such that

$$A^{-1} \approx X_U X_L P \quad \text{with } \|X_L P A X_U - I\| \le \varepsilon_{\rm tol},$$

which can directly be applied to calculating the determinant of A. Thus we adopt the accurate inverse LU factorization for this purpose.

In the algorithm for the accurate inverse LU factorization, we need an algorithm for accurately computing matrix multiplication (and therefore dot products), more precisely, as if computed in k-fold working precision and then stored into m pieces of working precision floating-point numbers for $k \ge 2$ and $1 \le m \le k$: Let $A = \sum_{i=1}^{s} A_i$ and $B = \sum_{i=1}^{t} B_i$ with A_i and B_i be floating-point matrices. Then we need to obtain a matrix $C = \sum_{i=1}^{m} C_i$ with C_i being floating-point matrices such that

$$|A \cdot B - C| \le \mathcal{O}(\mathbf{u}^m)|A \cdot B| + \mathcal{O}(\mathbf{u}^k)|A| \cdot |B|. \tag{4}$$

Here $|X| := (|x_{ij}|)$ for a real matrix $X = (x_{ij})$. Such accurate dot product algorithms have been developed in (Ogita et al., 2005; Rump et al., 2008a; Rump et al., 2008b), and they are very fast.

Let $\widehat{L}$, $\widehat{U}$ and P be the exact LU factors such that $PA = \widehat{L}\widehat{U}$ with $\widehat{L}$ being unit lower triangular. Then the ill-conditionedness of A is usually reflected in the upper triangular matrix $\widehat{U}$, i.e. $\kappa(A) \approx \kappa(\widehat{U})$. So, if a good approximate (right) inverse X_U of $\widehat{U}$, i.e. satisfying

$$\|\widehat{U} X_U - I\| \le \varepsilon_1 < 1, \tag{5}$$

is obtained, then the condition number of AX_U is expected to be relatively small compared with $\kappa(A)$, e.g. $\kappa(AX_U) < \mathbf{u}^{-1}$. After preconditioning A by X_U, it becomes not so difficult to calculate an approximate (left) inverse X_L of $\widehat{L}$ satisfying

$$\|X_L PAX_U - I\| \leq \varepsilon_2 < 1. \tag{6}$$

Since $\kappa(L)$ is usually small, it is sufficient to obtain L and X_U satisfying

$$\|PAX_U - L\| \leq \varepsilon_{\mathrm{tol}} < 1. \tag{7}$$

Using an algorithm for calculating accurate inverse LU factors proposed in (Ogita, 2009), we can obtain X_L, X_U and P satisfying (6). So, we can utilize the algorithm. For details, see (Ogita, 2009).

If such accurate inverse LU factors are available, then we can develop an algorithm for calculating enclosures of the determinant for extremely ill-conditioned matrices, whose structure is similar to the Rump's algorithm (Algorithm 2):

ALGORITHM 3. *An accurate and verified enclosure of* $\det(A)$.

1. *Compute a unit lower triangular matrix L, an upper triangular matrix X_U and a permutation matrix P such that $\|PAX_U - L\| \leq \varepsilon_{\mathrm{tol}}$ by the algorithm in* (Ogita, 2009).
2. *Compute an approximate inverse X_L of L by some standard numerical algorithm with pure floating-point arithmetic.*
3. *Compute an accurate inclusion $[B]$ of $X_L PAX_U$ using some verified algorithm for accurately computing matrix multiplications satisfying (4).*
4. *Compute an enclosure of* $\det(X_U)$ *by interval arithmetic.*
5. *Construct Gershgorin's circles for $[B]$ and then compute an enclosure of* $\det([B])$ *by interval arithmetic.*
6. *Compute an enclosure of* $\det(P)\det([B])/\det(X_U)$ *by interval arithmetic.*

The main computational effort is needed for Steps 1 and 3. If we use the algorithms for accurate dot product presented in (Ogita et al., 2005; Rump et al., 2008a; Rump et al., 2008b), then the computational cost for Steps 1 and 3 becomes $\mathcal{O}(k^3n^3)$ flops, where k is the number of floating-point matrices to express X_U. Although Step 2 requires $\mathcal{O}(n^3)$ flops, it can be done by pure floating-point arithmetic. The cost for the other steps using interval arithmetic is almost negligible. Moreover, the computational cost for Step 1 strongly depends on accurate matrix multiplications. Namely, the faster an algorithm for accurate matrix multiplication in use is, the faster Algorithm 3 is.

REMARK 2. *One may compute inverse QR factors instead of the inverse LU ones, i.e. $\|Q^T AX_R - I\| \leq \varepsilon_{\mathrm{tol}}$, where X_R is an upper triangular matrix and Q is an approximately orthogonal matrix. Although the algorithm becomes more stable, it needs to compute an enclosure of* $\det(Q)$*, whose absolute value is approximately equal to one.*

4. Exact determinant

As mentioned before, an accurate enclosure of the determinant of a real matrix A can be obtained by Algorithm 3. Suppose A is an integer matrix. If a rigorous enclosure $[\underline{d}, \overline{d}]$ of $\det(A)$ such that

$$\underline{d} \le \det(A) \le \overline{d} \quad \text{with } \lceil \underline{d} \rceil = \lfloor \overline{d} \rfloor = d \tag{8}$$

is obtained, then it mathematically proves that $\det(A) = d$. Namely, the verified numerical computations can calculate the exact solution of the problem in this case.

However, if $|\det(A)| \ge \mathbf{u}^{-1}$, then it is not possible in general to represent $\det(A)$ by a floating-point number. If that is the case, we need to extend Algorithm 3, especailly the algorithm for inverse LU factorization. Therefore, we do not treat such a case in this paper.

The following is an algorithm for calculating the exact determinant for an integer matrix in case of $|\det(A)| < \mathbf{u}^{-1}$:

ALGORITHM 4. *Let A be an integer square matrix. If $\det(A) < \mathbf{u}^{-1}$, then the following algorithm returns $d = \det(A)$. Otherwise, it returns an enclosure $[\underline{d}, \overline{d}]$ of $\det(A)$.*

1. *Apply Algorithm 3 for obtaining an enclosure $[d] := [\underline{d}, \overline{d}]$ of $\det(A)$ with some tolerance $\varepsilon_{\rm tol}$.*
2. *If $\lceil \underline{d} \rceil = \lfloor \overline{d} \rfloor = d$, then return d. Otherwise, put $\hat{d} := \max\{|\underline{d}|, |\overline{d}|\}$.*
3. *If $\hat{d} < 0.1 \cdot \mathbf{u}^{-1}$, then set $\varepsilon_{\rm tol} := 0.1 \cdot \hat{d}^{-1}$ and return to Step 1. Otherwise, return $[d]$.*

Next, we consider the case where A is singular. For any matrices A_1 and A_2 satisfying

$$\begin{cases} A(:,j) &= A_1(:,j) + A_2(:,j) \quad \text{for any } j \\ A(:,k) &= A_1(:,k) = A_2(:,k) \quad \text{for all } k \ne j \end{cases},$$

it holds from the linearity of the determinant that $\det(A) = \det(A_1) + \det(A_2)$. In general, we can expand $\det(A)$ as $\det(A) = \sum_{\nu=1}^{p} \det(A_\nu)$ with $\det(A_\nu) \ne 0$ for $1 \le \nu \le p$.

Let us consider the following example on Matlab:

```
>> T = randi([-100,100],n,n-1); % pseudo-random integers between -100 to 100
>> A = [T, sum(T,2)]; % Generate a singular matrix A
A =
     4    53    57   -55   -14    98   143
   -89   -60   -32   -93   -50    51  -273
   -86   -24    66   -36    56   -87  -111
    97    20    29   -77   -86   -77   -94
   -35   -47   -85   -88    36    21  -198
   -74     3    14    33   -93    78   -39
    58   -75    93   -37    70    89   198
```

Since A is singular, the true determinant of A is equal to zero. However, Matlab's built-in function returns a wrong answer as follows:

```
>> det(A) % Matlab's built-in function based on LU facotorization
ans =
    -1
```

This is due to the rounding errors in LU factorization by floating-point arithmetic. Here, for example, we put A_1 and A_2 as follows:

```
>> A1=[[A(1,1); zeros(n-1,1)] A(:,2:end)], A2=[[0; A(2:end,1)] A(:,2:end)]
A1 =
     4    53    57   -55   -14    98   143
     0   -60   -32   -93   -50    51  -273
     0   -24    66   -36    56   -87  -111
     0    20    29   -77   -86   -77   -94
     0   -47   -85   -88    36    21  -198
     0     3    14    33   -93    78   -39
     0   -75    93   -37    70    89   198
A2 =
     0    53    57   -55   -14    98   143
   -89   -60   -32   -93   -50    51  -273
   -86   -24    66   -36    56   -87  -111
    97    20    29   -77   -86   -77   -94
   -35   -47   -85   -88    36    21  -198
   -74     3    14    33   -93    78   -39
    58   -75    93   -37    70    89   198
```

Using the linearity of the determimant and Algorithm 3, we have the following result:

```
>> d1 = exactdet(A1)  % Algorithm 4 for A1
d1 =
     2.363144475936000e+12
>> d2 = exactdet(A2)  % Algorithm 4 for A2
d2 =
    -2.363144475936000e+12
>> d1 + d2 % det(A)
ans =
     0
```

Thus we can prove $\det(A) = 0$ and thus the singularity of A in this case. However, there are two problems in this approach: One is that absolute values of $\det(A_1)$ and $\det(A_2)$ (`d1` and `d2` in the above example) are not necessarily small even if $\det(A) = 0$. If $|\det(A_1)| \geq \mathbf{u}^{-1}$, then the above-mentioned approach cannot simply apply for proving the singularity of A. It frequently happens for large dimensional problems, independent of the condition number of A. The other is that there is a possibility for $\det(A_1) = \det(A_2) = 0$. If that is the case, we further need to expand A_1 and A_2.

Table I. Average of relative errors of resultant intervals for the determinant of random matrices.

n	Algorithm 1	Algorithm 2	Algorithm 3
10	$3.23 \cdot 10^{-13}$	$1.26 \cdot 10^{-13}$	$3.02 \cdot 10^{-14}$
50	$3.13 \cdot 10^{-11}$	$1.63 \cdot 10^{-11}$	$1.70 \cdot 10^{-12}$
100	$1.93 \cdot 10^{-10}$	$1.38 \cdot 10^{-10}$	$1.16 \cdot 10^{-11}$
200	$2.03 \cdot 10^{-9}$	$2.03 \cdot 10^{-9}$	$9.37 \cdot 10^{-11}$

Table II. Relative error of a resultant interval for the determinant of scaled Hilbert matrix.

n	$\kappa(A)$	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	$\det(A)$
5	$4.77 \cdot 10^{5}$	$2.04 \cdot 10^{-10}$	$6.53 \cdot 10^{-11}$	$5.81 \cdot 10^{-15}$	0 (exact)	381024
8	$1.53 \cdot 10^{10}$	$1.30 \cdot 10^{-5}$	$2.32 \cdot 10^{-6}$	$1.59 \cdot 10^{-14}$	0 (exact)	778350798225
10	$1.60 \cdot 10^{13}$	$1.53 \cdot 10^{-2}$	$3.80 \cdot 10^{-3}$	$2.73 \cdot 10^{-14}$	$2.73 \cdot 10^{-14}$	$> \mathbf{u}^{-1}$
15	$6.12 \cdot 10^{20}$	≥ 1	≥ 1	$6.65 \cdot 10^{-14}$	$6.65 \cdot 10^{-14}$	$> \mathbf{u}^{-1}$
20	$2.45 \cdot 10^{28}$	≥ 1	≥ 1	$1.37 \cdot 10^{-13}$	$1.37 \cdot 10^{-13}$	$> \mathbf{u}^{-1}$

5. Numerical results

We present some numerical results. We use a PC with Intel Core 2 Duo 1.86 GHz CPU and Matlab R2009a with INTLAB 5.5 (Rump, 1999). All computations are done in IEEE 754 double precision.

First, we treat random matrices with several n, whose elements are pseudo-random values drawn from the standard normal distribution. To see the accuracy of the results, we measure the relative error of the resultant interval $[d] \ni \det(A)$ by

$$\frac{\mathrm{rad}[d]}{|\mathrm{mid}[d]|},$$

where $\mathrm{mid}[d] := (\underline{d} + \overline{d})/2$ and $\mathrm{rad}[d] := (\overline{d} - \underline{d})/2$. For each n, we generate 50 samples and take the average of the relative errors. The results are displayed in Table I.

In these examples, we do not see much difference among all the algorithms tested since the matrices are not so ill-conditioned. In fact, the condition numbers of the matrices are around 10^2.

Next, we treat Hilbert matrix, which is widely known as an ill-conditioned matrix. To avoid rounding errors for an input matrix, we generate scaled Hilbert matrix whose condition number is the same as the original Hilbert matrix. Then the scaled Hilbert matrix is an integer matrix whose elements are exactly representable in double precision floating-point numbers for $n \leq 21$. So, we also test Algorithm 4 that attempts to compute the exact determinant. The results are displayed in Table II.

In the cases of $n = 15$ and $n = 20$, the condition number of A is greater than $\mathbf{u}^{-1} \approx 9.0 \cdot 10^{15}$. Then Algorithm 1 nor 2 cannot give meaningful results, while Algorithms 3 and 4 can. We confirm that Algorithm 4 can compute the exact determinant for $n \leq 8$.

Table III. Relative error and computing time for the determinant of Rump matrix with $n = 100$ and specified condition number. In all cases, $\det(A) = 1$.

	Algorithm 3 (verified)		Algorithm 4 (exact or verified)		SYM (exact)	GMP (non-verified)
$\kappa(A)$	Relative error	Time (sec)	Relative error	Time (sec)	Time (sec)	Time (sec)
$8.83 \cdot 10^{20}$	$5.34 \cdot 10^{-12}$	0.32	0 (exact)	0.33	24.38	0.40
$7.85 \cdot 10^{41}$	$5.69 \cdot 10^{-12}$	0.47	0 (exact)	0.47	59.55	0.92
$4.71 \cdot 10^{64}$	$5.22 \cdot 10^{-12}$	0.92	0 (exact)	0.92	76.08	2.15
$1.79 \cdot 10^{82}$	$5.78 \cdot 10^{-12}$	1.30	0 (exact)	1.30	161.48	2.15
$2.26 \cdot 10^{104}$	$4.77 \cdot 10^{-12}$	1.70	0 (exact)	1.70	183.82	2.15

Next, we compare computing times for our proposed algorithms to that for the symbolic computation (labeled by 'SYM') provided by Matlab as Symbolic Math Toolbox. Moreover, computing time for a standard numerical algorithm based on Gaussian elimination using a fast long precision library GMP 4.2.2 (GMP, 2007) (labeled by 'GMP') is also measured as the following trial-and-error algorithm:

```
function d = GMP_det_test(A,tol)
  prec = 106; % default precision
  d_old = 0;
  while 1
    d = GMP_det(A,prec); % MEX-function
    if abs(d_old - d) < tol*abs(d)
      break
    else
      prec = 2*prec;
      d_old = d;
    end
  end
```

Here the function `GMP_det` is compiled by MEX (Matlab external interfaces) with GNU Compiler Collection 4.4, so that it does not suffer from Matlab's interpretation overhead. Note that this algorithm does not guarantee the accuracy of the result. So, the computing time is just for reference.

To treat more ill-conditioned problems, we use the Rump's method (Rump, 1991) for generating extremely ill-conditioned matrices. It is implemented as `randmat(n,cnd)` in INTLAB, for which we can set a dimension `n` and an anticipated condition number `cnd`. We know the determinant of ill-conditioned matrices generated by `randmat` is equal to zero. In the following examples, we fix `n` $= 100$ and vary `cnd` from 10^{20} to 10^{100}. For the function `GMP_det`, we set `tol` $= 10^{-12}$. The results are displayed in Table III.

From the results, we can confirm that the computing times for the proposed algorithms (Algorithms 3 and 4) are much faster than that for Matlab's symbolic computations (SYM). Moreover, Algorithm 4 verifies that $\det(A) = 1$. Although the computing times for the proposed algorithms

Table IV. Relative error and computing time for the determinant of Rump matrix with $n = 500$ and specified condition number. In all cases, $\det(A) = 1$.

	Algorithm 3 (verified)		Algorithm 4 (exact or verified)	
$\kappa(A)$	Relative error	Time (sec)	Relative error	Time (sec)
$1.31 \cdot 10^{20}$	$5.57 \cdot 10^{-10}$	5.72	0 (exact)	6.12
$2.70 \cdot 10^{43}$	$5.45 \cdot 10^{-10}$	13.01	0 (exact)	13.38
$4.97 \cdot 10^{59}$	$5.78 \cdot 10^{-10}$	22.05	0 (exact)	22.53
$3.85 \cdot 10^{81}$	$5.48 \cdot 10^{-10}$	46.43	0 (exact)	46.47
$8.02 \cdot 10^{103}$	$6.03 \cdot 10^{-10}$	62.73	0 (exact)	64.73

are comparable to that for the GMP-based algorithm (GMP), the latter does not guarantee the result accuracy as mentioned before.

Finally, to treat ill-conditioned and larger dimensional problems, we again use the Rump matrix which appeared in the previous examples. In the following examples, we fix `n` $= 500$ and vary `cnd` from 10^{20} to 10^{100}. We measure the relative error of the result and the computing times for the proposed algorithms. The results are displayed in Table IV.

Regardless of the ill-conditionedness, Algorithm 3 gives verified and sufficiently accurate results. Moreover, Algorithm 4 verifies that $\det(A) = 1$. Algorithms 3 and 4 require reasonable computing times which correspond to the condition numbers.

6. Conclusion

We presented an algorithm (Algorithm 3) for calculating an accurate and verified enclosure of the matrix determinant. The algorithm can adaptively and efficiently treat extremely ill-conditioned cases by using an accurate inverse LU factorization. Next, we extend the algorithm to Algorithm 4 for calculating the exact determinant of integer matrices. The computational cost of the proposed algorithms depends on difficulty in solving a problem, i.e. a condition number of the input matrix. We confirmed the performance of the algorithms by some numerical results.

Acknowledgements

The author would like to express his sincere thanks to Prof. Shin'ichi Oishi from Waseda University for his stimulating discussions.

References

Brönnimann, H., C. Burnikel and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109(1–2):25–47, 2001.

Emiris, I. Z. and V. Y. Pan. Improved algorithms for computing determinants and resultants. *Journal of Complexity*, 21(1):43–71, 2005.

GMP: GNU Multiple Precision Arithmetic Library, version 4.2.2, 2007. Code and documentation available at `http://gmplib.org/` .

Higham, N. J. *Accuracy and Stability of Numerical Algorithms, 2nd ed.* SIAM, Philadelphia, PA, 2002.

Kaltofen, E. and G. Villard. Computing the sign or the value of the determinant of an integer matrix, a complexity survey. *J. Comput. Appl. Math.*, 162(1):133–146, 2004.

Lee, D. J. and I. C. F. Ipsen. Zone determinant expansions for nuclear lattice simulations. *Phys. Rev. C*, 68(6):064003, 2003.

Ogita, T. Accurate matrix factorization: Inverse LU and inverse QR factorizations. submitted for publication, 2009.

Ogita, T., S. M. Rump and S. Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.

Oishi, S., K. Tanabe, T. Ogita and S. M. Rump: Convergence of Rump's method for inverting arbitrarily ill-conditioned matrices. *J. Comp. Appl. Math.*, 205(1):533–544, 2007.

Pan, V. Y. and Y. Yu. Certification of numerical computation of the sign of the determinant of a matrix. *Algorithmica*, 30:708–724, 2001.

Rump, S. M. A class of arbitrarily ill-conditioned floating-point matrices. *SIAM J. Matrix Anal. Appl.*, 12(4):645–653, 1991.

Rump, S. M. Approximate inverses of almost singular matrices still contain useful information. *Forschungsschwerpunktes Informations- und Kommunikationstechnik*, Technical Report 90.1, Hamburg University of Technology, 1990.

Rump, S. M. Computer-assisted Proofs and Self-validating Methods. *Accuracy and Reliability in Scientific Computing* (Chapter 10), SIAM, Philadelphia, PA, 2005.

Rump, S. M. INTLAB – INTerval LABoratory. *Developments in Reliable Computing* (Tibor Csendes ed.), Kluwer Academic Publishers, Dordrecht, 1999, 77–104.

Rump, S. M. Inversion of extremely ill-conditioned matrices in floating-point. Japan J. Indust. Appl. Math., to appear.

Rump, S. M., T. Ogita and S. Oishi. Accurate floating-point summation Part I: Faithful rounding. *SIAM J. Sci. Comput.*, 31(1)189–224, 2008.

Rump, S. M., T. Ogita and S. Oishi. Accurate floating-point summation Part II: Sign, K-fold faithful and rounding to nearest. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.

Smagina, Ye. and I. Brewer. Using interval arithmetic for robust state feedback design. *Systems & Control Letters*, 46:187–194, 2002.

Smith, L. B. Interval arithmetic determinant evaluation and its use in testing for a Chebyshev system. Communications of the ACM, 12(2):89–93, 1969.