Front. Comput. Sci. China 2008, 2(1): 67–80
DOI 10.1007/s11704-008-0005-z

RESEARCH ARTICLE

Fraction-free matrix factors: new forms for LU and QR factors

Wenqin ZHOU (✉), David J. JEFFREY

Applied Mathematics Department, University of Western Ontario, London, Ontario, Canada

Abstract Gaussian elimination and LU factoring have been greatly studied from the algorithmic point of view, but much less from the point view of the best output format. In this paper, we give new output formats for fraction free LU factoring and for QR factoring. The formats and the algorithms used to obtain them are valid for any matrix system in which the entries are taken from an integral domain, not just for integer matrix systems. After discussing the new output format of LU factoring, the complexity analysis for the fraction free algorithm and fraction free output is given. Our new output format contains smaller entries than previously suggested forms, and it avoids the gcd computations required by some other partially fraction free computations. As applications of our fraction free algorithm and format, we demonstrate how to construct a fraction free QR factorization and how to solve linear systems within a given domain.

Keywords fraction free LU factoring, fraction free Gaussian elimination, fraction free QR factoring, complexity analysis

1 Introduction

Various applications including robot control and threat analysis have resulted in developing efficient algorithms for the solution of systems of polynomial and differential equations. This involves significant linear algebra sub-problems, which are not standard numerical linear algebra problems. The "arithmetic" that is needed is usually algebraic in nature and must be handled exactly [1]. In the standard numerical linear algebra approach, the cost of each operation during the standard Gaussian elimination is regarded as being the same, and the cost of numerical Gaussian elimination is obtained by counting the number of operations. For a $n \times n$ matrix with floating point numbers as entries, the cost of standard Gaussian elimination is $O(n^3)$.

However, in exact linear algebra problems, the cost of each operation during the standard Gaussian elimination may vary because of the growth of the entries [2]. For instance, in a polynomial ring over the integers, the problem of range growth manifests itself both in increased polynomial degrees and in the size of the coefficients. It is easy to modify the standard Gaussian elimination to avoid all division operations, but this leads to a very rapid growth [3–6].

One way to reduce this intermediate expression swell problem is the use of fraction free algorithms. Perhaps the first description of such an algorithm was due to Bareiss [7]. Afterwards, efficient methods for alleviating the expression swell problems from Gaussian elimination are given in Refs. [3,8–13].

Let us shift from Gaussian elimination to LU factoring. Turing's description of Gaussian elimination as a matrix factoring is well established in linear algebra [14]. The factoring takes different forms under the names Cholesky, Doolittle-LU, Crout-LU or Turing factoring. Only relatively recently has there been an attempt to combine LU factoring with fraction free Gaussian elimination [1,15]. One approach was given by Corless and Jeffrey in Ref. [15], see Theorem 2. In this paper, we use the standard fraction free Gaussian elimination algorithm given by Bareiss [7] and give a new fraction free LU factoring and a new fraction free QR factoring.

A procedure called fraction free LU decomposition already exists in MAPLE. Here is an example of its use.

Example 1 Let A be a 3×3 matrix with polynomials as entries,

$$A := \begin{bmatrix} x & 1 & 3 \\ 3 & 4 & 7 \\ 8 & 1 & 9 \end{bmatrix}.$$

If we use MAPLE to compute the fraction free LU factoring, we have

Received September 4, 2007; accepted December 10, 2007

E-mail: wzhou25@uwo.ca

>LUDecomposition(A, method = FractionFree);

$$\begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} \cdot A = \begin{bmatrix} 1 & & \\ 3/x & 1/x & \\ 8/x & \dfrac{x-8}{x(4x-3)} & \dfrac{1}{4x-3} \end{bmatrix} \cdot \begin{bmatrix} x & 1 & 3 \\ & 4x-3 & 7x-9 \\ & & 29x-58 \end{bmatrix}. \qquad (1)$$

The upper triangular matrix has polynomial entries. However, there are still fractions in the lower triangular matrix. In this paper, we call this factoring a partially fraction free LU factoring. We recall the corresponding algorithm below.

In addition, there are symbolic methods based on modular methods for solving systems with integer or polynomial coefficients exactly, such as p–adic lifting [16,17], computation modulo many different primes and using Chinese remainder theorem, or the recent high order lifting techniques [18,19], etc. However, the fraction free algorithms we introduce in this paper enable our computations over arbitrary integral domains. Moreover, at the complexity level, the new fraction free LU factoring algorithm can save a logarithmic factor over existing fraction free LU factoring algorithms.

Section 2 presents a format for LU factoring that is completely fraction free. In Section 3, we give a completely fraction free LU factoring algorithm and its time complexity, compared with the time complexity of a partially fraction free LU factoring. We show that partially fraction free LU factoring costs more than the completely fraction free LU factoring except for some special cases. Benchmarks follow in Section 4 and illustrate the complexity results from Section 3. The last part of the paper, Section 5, introduces the application of the completely fraction free LU factoring to obtain a similar structure for a fraction free QR factoring. In addition, it introduces the fraction free forward and backward substitutions to keep the whole computation in one domain for solving a linear system. Section 6 gives our conclusions.

2 Fraction free LU factoring

In 1968, Bareiss [7] pointed out that his integer-preserving Gaussian elimination (or fraction free Gaussian elimination) could reduce the magnitudes of the entries in the transformed matrices and increase the computational efficiency considerably in comparison with the corresponding standard Gaussian elimination. We also know that the conventional LU decomposition is used for solving several linear systems with the same coefficient matrix without the need to recompute the full Gaussian elimination. Here we combine these two ideas and give a new fraction free LU factoring.

In 1997, Nakos, Turner and Williams [1] gave an incompletely fraction free LU factorization. In the same year, Corless and Jeffrey [15] gave the following result on fraction free LU factoring.

Theorem 1 [Corless-Jeffrey] Any rectangular matrix $A \in \mathbb{Z}^{n\times m}$ may be written

$$F_1 PA = LF_2 U, \qquad (2)$$

where $F_1 = \mathrm{diag}(1, p_1, p_1p_2, \ldots, p_1p_2\ldots p_{n-1})$, P is a permutation matrix, $L \in \mathbb{Z}^{n\times n}$ is a unit lower triangular, $F_2 = \mathrm{diag}(1,1,p_1,p_1p_2,\ldots, p_1p_2\ldots p_{n-2})$, and $U \in \mathbb{Z}^{n\times m}$ are upper triangular matrices. The pivots p_i that arise are in $\mathbb{Z}$.

This factoring is modeled on other fraction free definitions, such as pseudo-division, and the idea is to inflate the given object or matrix so that subsequent divisions are guaranteed to be exact. However, although this model is satisfactory for pseudo-division, the above matrix factoring has two unsatisfactory features: firstly, two inflating matrices are required; and secondly, the matrices are clumsy, containing entries that increase rapidly in size. If the model of pseudo-division is abandoned, a tidier factoring is possible. This is the first contribution of this paper.

Theorem 2 Let $\mathbb{I}$ be an integral domain and $A = [a_{i,j}]_{i\leqslant n, j\leqslant m}$ be a matrix in $\mathbb{I}^{n\times m}$ with $n \leqslant m$ and such that the submatrix $[a_{i,j}]_{i,j\leqslant n}$ has full rank. Then, A may be written

$$PA = LD^{-1}U,$$

where

$$L = \begin{bmatrix} p_1 & & & & \\ L_{2,1} & p_2 & & & \\ \vdots & \vdots & \ddots & & \\ L_{n-1,1} & L_{n-1,2} & \cdots & p_{n-1} & \\ L_{n,1} & L_{n,2} & \cdots & L_{n,n-1} & 1 \end{bmatrix},$$

$$D = \begin{bmatrix} p_1 & & & & \\ & p_1p_2 & & & \\ & & \ddots & & \\ & & & p_{n-2}p_{n-1} & \\ & & & & p_{n-1} \end{bmatrix},$$

$$U = \begin{bmatrix} p_1 & U_{1,2} & \cdots & U_{1,n-1} & U_{1,n} & \cdots & U_{1,m} \\ & p_2 & \cdots & U_{2,n-1} & U_{2,n} & \cdots & U_{2,m} \\ & & \ddots & \vdots & \vdots & \vdots & \vdots \\ & & & p_{n-1} & U_{n-1,n} & \cdots & U_{n-1,m} \\ & & & & p_n & \cdots & U_{n,m} \end{bmatrix},$$

P is a permutation matrix, L and U are triangular as shown, and the pivots p_i that arise are in $\mathbb{I}$. The pivot p_n is also the determinant of the matrix $[a_{i,j}]_{i,j \leqslant n}$.

Proof For a full-rank matrix, there always exists a permutation matrix such that the diagonal pivots are non-zero during Gaussian elimination. Let P be such a permutation matrix for the full-rank matrix A (We will give details in the algorithm for finding this permutation matrix). Classical Gaussian elimination shows that PA admits the following factorization.

$$PA = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} & \cdots & a_{2,m} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} & \cdots & a_{3,m} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} & \cdots & a_{n,m} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & & & & \\ \frac{a_{2,1}}{a_{1,1}} & 1 & & & \\ \frac{a_{3,1}}{a_{1,1}} & \frac{a^{(1)}_{3,2}}{a^{(1)}_{2,2}} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ \frac{a_{n,1}}{a_{1,1}} & \frac{a^{(1)}_{n,2}}{a^{(1)}_{2,2}} & \frac{a^{(2)}_{n,3}}{a^{(2)}_{3,3}} & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & \cdots & a_{1,m} \\ & a^{(1)}_{2,2} & a^{(1)}_{2,3} & \cdots & a^{(1)}_{2,n} & \cdots & a^{(1)}_{2,m} \\ & & a^{(2)}_{3,3} & \cdots & a^{(2)}_{3,n} & \cdots & a^{(2)}_{3,m} \\ & & & \ddots & \vdots & \cdots & \vdots \\ & & & & a^{(n-1)}_{n,n} & \cdots & a^{(n-1)}_{n,m} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & & & & \\ \frac{a_{2,1}}{a_{1,1}} & 1 & & & \\ \frac{a_{3,1}}{a_{1,1}} & \frac{a^{(1)}_{3,2}}{a^{(1)}_{2,2}} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ \frac{a_{n,1}}{a_{1,1}} & \frac{a^{(1)}_{n,2}}{a^{(1)}_{2,2}} & \frac{a^{(2)}_{n,3}}{a^{(2)}_{3,3}} & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & & & & \\ & a^{(1)}_{2,2} & & & \\ & & a^{(2)}_{3,3} & & \\ & & & \ddots & \\ & & & & a^{(n-1)}_{n,n} \end{bmatrix}$$

$$\begin{bmatrix} 1 & \frac{a_{1,2}}{a_{1,1}} & \frac{a_{1,3}}{a_{1,1}} & \cdots & \frac{a_{1,n}}{a_{1,1}} & \cdots & \frac{a_{1,m}}{a_{1,1}} \\ & 1 & \frac{a^{(1)}_{2,3}}{a^{(1)}_{2,2}} & \cdots & \frac{a^{(1)}_{2,n}}{a^{(1)}_{2,2}} & \cdots & \frac{a^{(1)}_{2,m}}{a^{(1)}_{2,2}} \\ & & 1 & \cdots & \frac{a^{(2)}_{3,n}}{a^{(2)}_{3,3}} & \cdots & \frac{a^{(2)}_{3,m}}{a^{(2)}_{3,3}} \\ & & & \ddots & \vdots & \cdots & \vdots \\ & & & & 1 & \cdots & \frac{a^{(n-1)}_{n,m}}{a^{(n-1)}_{n,n}} \end{bmatrix}$$

$$= \ell \cdot d \cdot u.$$

The coefficients $a^{(k)}_{i,j}$ are related by the following relations:

$$i=1,\quad a_{i,j}^{(0)}=a_{i,j};$$

$$2\leqslant i\leqslant j\leqslant m,\quad a_{i,j}^{(i-1)}=\begin{vmatrix} a_{i-1,i-1}^{(i-2)} & a_{i-1,j}^{(i-2)} \\ \dfrac{a_{i,i-1}^{(i-2)}}{a_{i-1,i-1}^{(i-2)}} & \dfrac{a_{i,j}^{(i-2)}}{a_{i-1,i-1}^{(i-2)}} \end{vmatrix};$$

$$n\geqslant i\geqslant j\geqslant 2,\quad a_{i,j}^{(j-1)}=\begin{vmatrix} a_{j-1,j-1}^{(j-2)} & a_{j-1,j}^{(j-2)} \\ \dfrac{a_{i,j-1}^{(j-2)}}{a_{j-1,j-1}^{(j-2)}} & \dfrac{a_{i,j}^{(j-2)}}{a_{j-1,j-1}^{(j-2)}} \end{vmatrix}.$$

Let us define $A_{i,j}^{(k)}$ by $A_{i,j}^{(k)}=\begin{vmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} & a_{1,j} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} & a_{2,j} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k} & a_{k,j} \\ a_{i,1} & a_{i,2} & \cdots & a_{i,k} & a_{i,j} \end{vmatrix}.$

Hence, these are in $\mathbb{I}$.

We have the relations $A_{i,j}^{(k)}=\dfrac{1}{A_{k-1,k-1}^{(k-2)}}\begin{vmatrix} A_{k,k}^{(k-1)} & A_{k,j}^{(k-1)} \\ A_{i,k}^{(k-1)} & A_{i,j}^{(k-1)} \end{vmatrix},$

which was proved in Ref. [7]. From the definitions, we have

$$m\geqslant j\geqslant i\geqslant 1,\quad a_{i,j}^{(i-1)}=\frac{1}{A_{i-1,i-1}^{(i-2)}}A_{i,j}^{(i-1)},$$
$$n\geqslant j\geqslant i\geqslant 1,\quad a_{j,i}^{(i-1)}=\frac{1}{A_{i-1,i-1}^{(i-2)}}A_{j,i}^{(i-1)}.$$

So

$$\begin{aligned} \ell_{i,k}&=\frac{1}{a_{k,k}^{(k-1)}}\cdot a_{i,k}^{(k-1)} \\ &=\frac{A_{k-1,k-1}^{(k-2)}}{A_{k,k}^{(k-1)}}\cdot\frac{A_{i,k}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}} \\ &=\frac{A_{i,k}^{(k-1)}}{A_{k,k}^{(k-1)}};\qquad \ell_{k,k}=1; \\ u_{k,j}&=\frac{1}{a_{k,k}^{(k-1)}}\cdot a_{k,j}^{(k-1)} \\ &=\frac{A_{k-1,k-1}^{(k-2)}}{A_{k,k}^{(k-1)}}\cdot\frac{A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}} \\ &=\frac{A_{k,j}^{(k-1)}}{A_{k,k}^{(k-1)}};\qquad u_{k,k}=1; \\ d_{k,k}&=a_{k,k}^{(k-1)}=\frac{A_{k,k}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}}. \end{aligned}$$

Then, the fraction free LU form $PA=LD^{-1}U$ can be written as:

$$\begin{aligned} L_{i,k}&=A_{i,k}^{(k-1)}, && n\geqslant i\geqslant k\geqslant 1, \\ U_{k,j}&=A_{k,j}^{(k-1)}, && m\geqslant j\geqslant k\geqslant 1, \\ D_{k,k}&=\frac{A_{k-1,k-1}^{(k-2)}}{A_{k,k}^{(k-1)}}\cdot A_{k,k}^{(k-1)}\cdot A_{k,k}^{(k-1)} \\ &=A_{k-1,k-1}^{(k-2)}A_{k,k}^{(k-1)}, && n\geqslant k\geqslant 1, \end{aligned}$$

which is also equivalent to $PA=LD^{-1}U$, where

$$L=\begin{bmatrix} A_{1,1}^{(0)} & & & & \\ A_{2,1}^{(0)} & A_{2,2}^{(1)} & & & \\ \vdots & \vdots & \ddots & & \\ A_{n-1,1}^{(0)} & A_{n-1,2}^{(1)} & \cdots & A_{n-1,n-1}^{(n-2)} & \\ A_{n,1}^{(0)} & A_{n,2}^{(1)} & \cdots & A_{n,n-1}^{(n-2)} & 1 \end{bmatrix},$$

$$D=\begin{bmatrix} A_{1,1}^{(0)} & & & & \\ & A_{1,1}^{(0)}A_{2,2}^{(1)} & & & \\ & & \ddots & & \\ & & & A_{n-2,n-2}^{(n-3)}A_{n-1,n-1}^{(n-2)} & \\ & & & & A_{n-1,n-1}^{(n-2)} \end{bmatrix},$$

$$U=\begin{bmatrix} A_{1,1}^{(0)} & A_{1,2}^{(0)} & \cdots & A_{1,n-1}^{(0)} & A_{1,n}^{(0)} & \cdots & A_{1,m}^{(0)} \\ & A_{2,2}^{(1)} & \cdots & A_{2,n-1}^{(1)} & A_{2,n}^{(1)} & \cdots & A_{2,m}^{(1)} \\ & & \ddots & \vdots & \vdots & \vdots & \vdots \\ & & & A_{n-1,n-1}^{(n-2)} & A_{n-1,n}^{(n-2)} & \cdots & A_{n-1,m}^{(n-2)} \\ & & & & A_{n,n}^{(n-1)} & \cdots & A_{n,m}^{(n-1)} \end{bmatrix}.$$

□

Example 2 Let us compute the fraction free LU factoring of the same matrix A as in Example 1 (here, we take P to be the identity matrix).

$$A:=\begin{bmatrix} x & 1 & 3 \\ 3 & 4 & 7 \\ 8 & 1 & 9 \end{bmatrix};\quad P=\begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}.$$

In Example 1, we found that there were still fractions in the fraction free LU factoring from the MAPLE computation. When using our new LU factoring, there is no fraction appearing in the upper or lower triangular matrix.

$$P\cdot A=\begin{bmatrix} x & & \\ 3 & 4x-3 & \\ 8 & x-8 & 1 \end{bmatrix}\cdot\begin{bmatrix} x & & \\ & x(4x-3) & \\ & & 4x-3 \end{bmatrix}^{-1}\cdot\begin{bmatrix} x & 1 & 3 \\ & 4x-3 & 7x-9 \\ & & 29x-58 \end{bmatrix}$$

This new output is better than the old one for the following two aspects: first, this new LU factoring form keeps the computation in the same domain; second, the division used in the new factoring is an exact division, while in Example 1 fraction free LU factoring the division needs gcd computations for the lower triangular matrix as in Eq. 1. We give a more general comparison of these two forms on their time complexity in Theorems 5 of Section 3.

3 Completely fraction free algorithm and time complexity analysis

Here we give an algorithm for computing a completely fraction free LU factoring (CFFLU). This is generic code; an actual MAPLE implementation would make additional optimizations with respect to different input domains.

Algorithm 1 Completely Fraction free LU factoring (CFFLU)

Input: A $n \times m$ matrix A, with $m \geqslant n$.

Output: Four matrices P, L, D, U, where P is a $n \times n$ permutation matrix, L is a $n \times n$ lower triangular matrix, D is a $n \times n$ diagonal matrix, U is a $n \times m$ upper triangular matrix and $PA = LD^{-1}U$.

```
U := Copy(A); (n,m) := Dimension(U): oldpivot := 1;
L:=IdentityMatrix(n,n, 'compact'=false);
DD:=ZeroVector(n, 'compact'=false);
P := IdentityMatrix(n, n, 'compact'=false);
for k from 1 to n−1 do
   if U[k,k] = 0 then
      kpivot := k+1;
      Notfound := true;
      while kpivot < (n+1) and Notfound do
         if U[kpivot, k] <> 0 then
            Notfound := false;
         else
            kpivot := kpivot +1;
         end if;
      end do:
      if kpivot = n+1 then
         error "Matrix is rank deficient";
      else
         swap := U[k, k..n];
         U[k,k..n] := U[kpivot, k..n];
         U[kpivot, k..n] := swap;
         swap := P[k, k..n];
         P[k, k..n] := P[kpivot, k..n];
         P[kpivot, k..n] := swap;
      end if:
   end if:
   L[k,k]:=U[k,k];
   DD[k] := oldpivot * U[k, k];
   Ukk := U[k,k];
   for i from k+1 to n do
      L[i,k] := U[i,k];
      Uik := U[i,k];
      for j from k+1 to m do
         U[i,j]:=normal((Ukk*U[i,j]−U[k,j]*Uik)/oldpivot);
      end do;
      U[i,k] := 0;
   end do;
   oldpivot:= U[k,k];
end do;
DD[n]:= oldpivot;                                              □
```

For comparison, we also recall a partially fraction free LU factoring (PFFLU).

Algorithm 2 Partially Fraction free LU factoring (PFFLU)

Input: A $n \times m$ matrix A.

Output: Three matrices P, L and U, where P is a $n \times n$ permutation matrix, L is a n $\times$ n lower triangular matrix, U is a $n \times m$ fraction free upper triangular matrix and $PA = LU$.

```
U := Copy(A); (n,m) := Dimension(U): oldpivot := 1;
L:=IdentityMatrix(n,n, 'compact'=false);
P := IdentityMatrix(n, n, 'compact'=false);
for k from 1 to n−1 do
   if U[k,k] = 0 then
      kpivot := k+1;
      Notfound := true;
      while kpivot < (n+1) and Notfound do
         if U[kpivot, k] <> 0 then
            Notfound := false;
         else
            kpivot := kpivot +1;
         end if;
      end do:
      if kpivot = n+1 then
         error "Matrix is rank deficient";
      else
         swap := U[k, k..n];
         U[k,k..n] := U[kpivot, k..n];
         U[kpivot, k..n] := swap;
         swap := P[k, k..n];
         P[k, k..n] := P[kpivot, k..n];
         P[kpivot, k..n] := swap;
      end if:
```

```
    end if:
    L[k,k]:=1/oldpivot;
    Ukk := U[k,k];
    for i from k+1 to n do
      L[i,k] := normal(U[i,k]/(oldpivot * U[k, k]));
      Uik := U[i,k];
      for j from k+1 to m do
        U[i,j]:=normal((Ukk*U[i,j]−U[k,j]*Uik)/oldpivot);
      end do;
      U[i,k] := 0;
    end do;
    oldpivot:= U[k,k];
  end do;
  L[n,n]:= 1/oldpivot;
```
□

The main difference between Algorithm 1 and Algorithm 2 is that Algorithm 2 uses non-exact divisions when computing the L matrix. The reason we give these two algorithms is that we want to show the advantage of a fraction free output format.

Theorem 3 Let A be a $n \times m$ matrix of full rank with entries in a domain $\mathbb{I}$ and $n \leqslant m$. On input A, Algorithm 1 outputs four matrices P, L, D, U with entries in $\mathbb{I}$ such that $PA = LD^{-1}U$, P is a $n \times n$ permutation matrix, L is a $n \times n$ lower triangular matrix, D is a $n \times n$ diagonal matrix, U is a $n \times m$ upper triangular matrix. Furthermore, all divisions are exact.

Proof In Algorithm 1, each pass through the main loop starts by finding a non-zero pivot, and reorders the row accordingly. For the sake of proof, we can suppose that the rows have been permuted from the start, so that no permutation is necessary.

Then we prove by induction that at the end of step k, for $k = 1,\ldots,n-1$, we have

- $QD[1] = A^{(0)}_{1,1}$ and $D[i] = A^{(i-2)}_{i-1,i-1}A^{(i-1)}_{i,i}$ for $i = 1,\ldots,k$,
- $L[i,j] = A^{(j-1)}_{i,j}$ for $j = 1,\ldots,k$ and $i = 1,\ldots,n$,
- $U[i,j] = A^{(i-1)}_{i,j}$ for $i = 1,\ldots,k$ and $j = i,\ldots,m$,
- $U[i,j] = A^{(k)}_{i,j}$ for $i = k+1,\ldots,n$ and $j = k+1,\ldots,m$,
- all other entries are 0.

These equalities are easily checked for $k = 1$. Suppose that this holds at step k, and let us prove it at step $k+1$. Then,

- for $i = k+1,\ldots,n$, $L[i,k+1]$ gets the value $U[i,k+1] = A^{(k)}_{i,k+1}$,
- $D[k+1]$ gets the value $A^{(k-1)}_{k,k}A^{(k)}_{k+1,k+1}$,
- for $i,j = k+2,\ldots,m$, $U[i,j]$ gets the value
$$\frac{A^{(k-1)}_{k,k}A^{(k-1)}_{i,j} - A^{(k-1)}_{k,j}A^{(k-1)}_{i,k}}{A^{(k-2)}_{k-1,k-1}} = A^{(k)}_{i,j},$$
- $U[i,k+1]$ gets the value 0 for $i = k+2,\ldots,n$.

This proves our statement by induction. In particular, as claimed, all divisions are exact. □

In the following part of this section, we discuss the advantages of the completely fraction free LU factoring and give the time complexity analysis for the fraction free algorithms and fraction free outputs in matrices with univariate polynomial entries and with integer entries. First, let us introduce two definitions about a length of an integer and a multiplication time (Ref. [20], §2.1, §8.3).

Definition 1 For a nonzero integer $a \in \mathbb{Z}$, we define the length $\lambda(a)$ of a as

$$\lambda(a) = \lfloor \log|a| + 1 \rfloor,$$

where $\lfloor . \rfloor$ denotes rounding down to the nearest integer.

Definition 2 Let $\mathbb{I}$ be a ring (commutative, with 1). We call a function $M : \mathbb{N}_{>0} \to \mathbb{R}_{>0}$ a multiplication time for $\mathbb{I}[x]$ if polynomials in $\mathbb{I}[x]$ of degree less than n can be multiplied using at most $M(n)$ operations in $\mathbb{I}$, and if M is such that $M(n)+M(m) \leqslant M(n+m)$ holds for all n, m. Similarly, a function M as above is called a multiplication time for $\mathbb{Z}$ if two integers of length n can be multiplied using at most $M(n)$ word operations, and if the superlinearity condition given above holds.

All known multiplication algorithm lead to a multiplication time. Table 1 summarizes the multiplication times for some general algorithms.

Table 1 Various polynomial multiplication algorithms and their running times

algorithm	$M(n)$
classical	$O(n^2)$
Karatsuba(Karatsuba & Ofman 1962)	$O(n^{1.59})$
FFT multiplication (provided that $\mathbb{I}$ supports the FFT)	$O(n \log n)$
Schönhage & Strassen (1971), Schönhage (1977)	
Cantor & Kaltofen (1991); FFT based	$O(n \log n \log\log n)$

The cost of dividing two integers of length ℓ is $O(M(\ell))$ word operations, and the cost of a gcd computation is $O(M(\ell)\log\ell)$ word operations. For two polynomials $a, b \in \mathbb{K}[x]$, where $\mathbb{K}$ is an arbitrary field, of degrees less than d, the cost of division is $O(M(d))$, and the cost of gcd computation is $O(M(d)\log d)$. For two univariate polynomials $a, b \in \mathbb{Z}[x]$ of degree less than d and coefficient's length bounded by ℓ, if a divides b and if the quotient has coefficients of length bounded by ℓ', the cost of division is $O(M(d(\max(\ell,\ell') + \log d)))$. If the division is non-exact, i.e., we need to compute the gcd of a and b, the cost is [20].

Lemma 1 From Ref. [21], for a $k \times k$ matrix $A = [a_{i,j}]$, with $a_{i,j} \in \mathbb{Z}[x_1, \ldots, x_m]$ with degree and length bounded by d and ℓ respectively, the degree and length of det(A) are bounded by kd and $k(\ell + \log k + d \log(m+1))$ respectively.

Based on the completely fraction free LU factoring Algorithm 1, at each kth step of fraction free Gaussian elimination, we have to compute

$$U_{k,j} = A_{k,j}^{(k-1)},$$

where

$$A_{k,j}^{(k-1)} = \det \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k-1} & a_{1,j} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k-1} & a_{2,j} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{k-1,1} & a_{k-1,2} & \cdots & a_{k-1,k-1} & a_{k-1,j} \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k-1} & a_{k,j} \end{pmatrix}_{k \times k}, \quad (3)$$

and

$$L_{i,k} = A_{i,k}^{(k-1)}, \; D_{k,k} = A_{k-1,k-1}^{(k-2)} A_{k,k}^{(k-1)}.$$

Lemma 2 If every entry $a_{i,j}$ of the matrix $A = [a_{i,j}]_{n \times n}$ is a univariate polynomial over a field $\mathbb{K}$ with degree less than d, we have $\deg\left(A_{i,j}^{(k)}\right) \leqslant kd$.

If every entry $a_{i,j}$ of the matrix $A = [a_{i,j}]_{n \times n}$ is in $\mathbb{Z}$ and has length bounded by ℓ, we have $\lambda\left(A_{i,j}^{(k)}\right) \leqslant k(\ell + \log k)$.

If every entry of the matrix $A = [a_{i,j}]n \times n$ is a univariate polynomial over $\mathbb{Z}[x]$ with degree less than d and coefficient's length bounded by ℓ, we have $\deg\left(A_{i,j}^{(k)}\right) \leqslant kd$ and $\lambda\left(A_{i,j}^{(k)}\right) \leqslant k(\ell + \log k + d \log 2)$.

Proof If $a_{i,j} \in \mathbb{Z}[x]$ has degree less than d, from Lemma 1, we have $\deg\left(A_{i,j}^{(k)}\right) \leqslant kd$. If $a_{i,j} \in \mathbb{Z}$ has length bounded by ℓ, from Eq. 3 and Lemma 1 with $m = 0$, we have $\lambda\left(A_{i,j}^{(k)}\right) \leqslant k(\ell + \log k)$. If $a_{i,j} \in \mathbb{Z}[x]$ has degree less than d and coefficient's length bounded by ℓ, from Eq. 3 and Lemma 1 with $m = 1$, we have $\deg\left(A_{i,j}^{(k)}\right) \leqslant kd$ and $\lambda\left(A_{i,j}^{(k)}\right) \leqslant k(\ell + \log k + d \log 2)$. □

In the following part of this section, we want to demonstrate that the difference between fraction free LU factoring and our completely fraction free LU factoring is the divisions used in computing their lower triangular matrices L. We discuss here only three cases. In case 1, we will analyze the cost of two algorithms with $A \in \mathbb{K}[x]$, where $\mathbb{K}$ is a field, i.e., we only consider the growth of degree during the factoring. In case 2, we will analyze the cost of two algorithms with $A \in \mathbb{Z}$, i.e. we only consider the growth of length during the factoring. In case 3, we will analyze the cost of both algorithms with $A \in \mathbb{Z}[x]$. For more cases, such as $A \in \mathbb{Z}[x_1, \ldots, x_m]$, the basic idea will be the same as these three basic cases.

Theorem 4 For a matrix $A = [a_{i,j}]_{n \times n}$ with entries in $\mathbb{K}[x]$, if every $a_{i,j}$ has degree less than d, the time complexity of completely fraction free LU factoring for A is bounded by $O(n^3 M(nd))$ operations in $\mathbb{K}$.

For a matrix $A = [a_{i,j}]_{n \times n}$ with entries in $\mathbb{Z}$, if every $a_{i,j}$ has length bounded by ℓ, the time complexity of completely fraction free LU factoring for A is bounded by $O\left(n^3 M(n \log n + n\ell)\right)$ *word operations.*

For a matrix $A = [a_{i,j}]_{n \times n}$ with univariate polynomial entries in $\mathbb{Z}[x]$, if every $a_{i,j}$ has degree less than d and has length bounded by ℓ, the time complexity of completely fraction free LU factoring for A is bounded by $O\left(n^3 M\left(n^2 d\ell + nd^2\right)\right)$ word operations.

Proof Let case 1 be the case $a_{i,j} \in \mathbb{K}[x]$ with $d = \max_{i,j} \deg(a_{i,j}) + 1$, case 2 be the case $a_{i,j} \in \mathbb{K}$ with $\ell = \max_{i,j}^{\lambda}(a_{i,j})$ and case 3 be the case $a_{i,j} \in \mathbb{Z}[x]$ with $d = \max_{i,j} \deg(a_{i,j}) + 1$ and $\ell = \max_{i,j}^{\lambda}(a_{i,j})$. From Lemma 2, at each step k, $\deg\left(A_{i,j}^{(k)}\right) \leqslant kd$ in case 1 and $\lambda\left(A_{i,j}^{(k)}\right) \leqslant k(\ell + \log k)$ in case 2, and $\lambda\left(A_{i,j}^{(k)}\right) \leqslant k(\ell + \log k + d \log 2)$ and $\deg\left(A_{i,j}^{(k)}\right) \leqslant kd$ in case 3.

When we do completely fraction free LU factoring, at the kth step, we have $(n-k)^2$ entries to compute. For each new entry from step $k-1$ to step k, we need to do at most two multiplication, one subtraction and one division. The cost will be bounded by $O(M(kd))$ for case 1, by $O(M(k(\ell + \log k)))$ for case 2 and by $O(M(kd(((2k)\ell + \log(2k) + d \log 2) + \log(kd))))$ for case 3.

Let c_1, c_2 and c_3 be constants, such that the previous estimates are bounded by

$$c_1 \times M(kd),$$

$$c_2 \times M(k(\ell + \log k)),$$

and

$$c_3 \times M\left(k^2 d\ell + kd \log k + kd^2 \log 2 + kd \log(kd)\right).$$

For case 1, the total cost for the completely fraction free LU factoring will be bounded by

$$\sum_{k=1}^{n-1} c_1 (n-k)^2 M(kd) = O\left(n^3 M(nd)\right).$$

For case 2, the total cost for the completely fraction free LU factoring will be bounded by

$$\sum_{k=1}^{n-1} c_2(n-k)^2 M(k(\ell+\log k)) = O\big(n^3 M(n\log n + n\ell)\big).$$

For case 3, the total cost for the completely fraction free LU factoring will be bounded by

$$\sum_{k=1}^{n-1} c_3(n-k)^2 M\big(2k^2 d\ell + kd\log(2k) + kd^2\log 2 + kd\log(kd)\big) = O\big(n^3 M\big(n^2 d\ell + nd^2\big)\big).$$

□

The extra gcd computation in Algorithm 2 when computing $L[i, k]{:} = U[i, k]/(\text{oldpivot}^* U[k, k])$ makes the fraction free LU factoring more expensive, as shown in the following theorem.

Theorem 5 For a matrix $A = [a_{i,j}]_{n\times n}$, if every $a_{i,j}\in\mathbb{K}[x]$ has degree less than d, the time complexity of the partially fraction free LU factoring for A is bounded by $O((n^2\log(nd)+n^3)M(nd))$.

For a matrix $A=[a_{i,j}]_{n\times n}$, if every $a_{i,j}\in\mathbb{Z}$ has length bounded by ℓ, the time complexity of partially fraction free LU factoring for A is bounded by $O\big(\big(n^2\log(n\log n+n\ell)+n^3\big)M(n\log n+n\ell)\big)$.

For a matrix $A=[a_{i,j}]_{n\times n}$, if every $a_{i,j}\in\mathbb{Z}[x]$ has degree less than d and coefficient length bounded by ℓ, the time complexity of the partially fraction free LU factoring for A is bounded by $O\big(n^3 M\big(n^2 d\ell+nd^2\big)+n^2\delta M\big)$, where $\delta M = M(nd)\log(nd)(n(\ell+\log n+d))M(\log(nd(\ell+d)))\log\log(nd(\ell+d))+ndM(n(\ell+\log n+d)\log(nd(\ell+d)))\log(n(\ell+d))$.

Proof As above, case 1 is $a_{i,j}\in\mathbb{K}[x]$ with $d=\max_{i,j}\deg(a_{i,j})+1$, case 2 is $a_{i,j}\in\mathbb{Z}$ with $\ell=\max^{\lambda}_{i,j}\big(a_{i,j}\big)$ and case 3 is $a_{i,j}\in\mathbb{Z}[x]$ with $d=\max_{i,j}\deg(a_{i,j})+1$ and $\ell=\max^{\lambda}_{i,j}\big(a_{i,j}\big)$. From Lemma 2, at each step kth, $\deg\Big(A^{(k)}_{i,j}\Big)\leqslant kd$ in case 1, $\lambda\Big(A^{(k)}_{i,j}\Big)\leqslant k(\ell+\log k)$ in case 2, and $\lambda\Big(A^{(k)}_{i,j}\Big)\leqslant k(\ell+\log k+d\log 2)$ and $\deg\Big(A^{(k)}_{i,j}\Big)\leqslant kd$ in case 3.

When we do fraction free LU factoring, at the kth step, we need to do the same computation for U matrix as in Algorithm 1. In addition, we have $n-k$ entries, which are the entries of kth column of L matrix, containing gcd computation for its division in order to get the common factors out of division.

For case 1, the total cost for the partially fraction free LU factoring will be bounded by

$$\sum_{k=1}^{n-1}\big(c_1(n-k)^2 M(kd)+c_2(n-k)M(kd)\log(kd)\big) = O\big(\big(n^2\log(nd)+n^3\big)M(nd)\big),$$

where c_1 and c_2 are constants.

For case 2, the total cost for the partially fraction free LU factoring will be bounded by

$$\sum_{k=1}^{n-1}\big(c_3(n-k)^2 M(k(\ell+\log k)) + c_4(n-k)M(k(\ell+\log k))\log(k(\ell+\log k))\big) = O((n^2\log(n\log n+n\ell)+n^3)M(n\log n+n\ell)),$$

where c_3 and c_4 are constants.

For case 3, after a few simplification the total cost for the partially fraction free LU factoring will be bounded by

$$O\big(n^3 M\big(n^2 d\ell+nd^2\big)+n^2\delta M\big),$$

where

$$\delta M = (n(\ell+\log n+d))\log(nd)\log\log(nd(\ell+d)) \\ M(nd)M(\log(nd(\ell+d)))+nd\log(n(\ell+d)) \\ M(n(\ell+\log n+d)\log(nd(\ell+d))).$$

□

Comparing Theorem 4 with Theorem 5, we see that partially fraction free LU factoring costs a little more than our completely fraction free LU. More precisely, consider for instance the case of polynomial matrices, for fixed degree d, both algorithms have complexity $O(n^3M(n))$. However, when n is fixed, the partially fraction free LU factoring algorithm has complexity $O(M(d)\log(d))$, where ours features a better bound $O(M(d))$. The behavior is the same for the other classes of matrices and will be confirmed by our benchmarks.

4 Benchmarks

We compare Algorithm 1 and Algorithm 2 on matrices with integer entries and univariate polynomial entries. We do not give a benchmark for matrices with multivariate polynomial entries because the running time is too long for the matrix whose size is larger than ten. We also give measure the time of the MAPLE command for fraction free LU. Because the implementation details are not published for the Maple command we use, we will only use it for our references.

As we know, the main difference in Algorithm 1 and Algorithm 2 is in their divisions. In our codes, we use the divide command for exact division in univariate polynomial case and the iquo command for integer case instead of the normal command. All results are obtained using the TTY version of MAPLE 10, running on an 2.8 Ghz Intel P4 with 1024 Megs of memory running Linux, and with time limit set to 2000 seconds for each fraction-free factoring operation, and garbage collection "frequency" (gcfreq) set to 2×10^7 bytes.

In the following tables, we label the time used by Algorithm 1 as CFFLU, the time used by Algorithm 2 as PFFLU and the time used by the Maple command LUDecomposition(A, method = FractionFree) as MapleLU. We also denote the size of a matrix as n and the length of an integer entry of a matrix as ℓ.

Table 2 gives the times of three algorithms on different sizes of matrices with integer entries of fixed length (Fig. 1). It also gives the relations between the logarithm of the time of completely fraction free LU factoring with the logarithm of the size of matrix (Fig. 2). If we use the Maple command Fit($a+b \cdot t,x,y,t$) to fit it, we find a slope equal to 4.335. This tells us that the relation between the time used by completely fraction free LU factoring and the size of the matrix is $t = O(n^{4.335})$. In the view of Theorem 4, this suggest that $M(n \log n)$ behaves like $O(n^{1.335})$ for integer multiplication in our range. This subquadratic behavior is consistent with the fact that MAPLE uses the GMP library for long integers multiplication. We also could use block operations instead of explicit loops in our codes: this is an efficient method in MATLAB, but it will be 10 times slower in MAPLE.

Table 2 Timings for fraction free LU factoring of random integer matrices generated by RandomMatrix(n,n, generator = $-10^{100} \ldots 10^{100}$)

n	CFFLU	PFFLU	MapleLU
10	1.00E-02	1.40E-02	2.80E-02
20	0.133	0.188	0.263
30	0.735	0.959	1.375
40	2.617	4.127	4.831
50	6.94	8.622	13.182
60	15.672	19.574	31.342
70	32.003	35.938	61.974
80	56.598	65.553	117.111
90	97.881	110.931	208.665
100	155.757	171.989	335.407
110	237.916	264.304	531.193
120	351.737	396.445	803.898
130	512.428	562.352	1184.575
140	709.42	775.139	1684.698
150	975.429	1060.02	> 2000
160	1298.419	1410.204	> 2000
170	1714.003	1868.879	> 2000

For fixed-size matrices with integer entries, we have the following two tables. Table 3 gives the times of three algorithms on matrices with different length of integer entries (Fig. 3). It gives us the relations between the logarithm of the time of completely fraction free LU factoring with the logarithm of the length of integer entry (Fig. 4). If we use the MAPLE command Fit($a+b \cdot t,x,y,t$) to fit it, we find a slope equal to 1.265. This tells us that the relation between

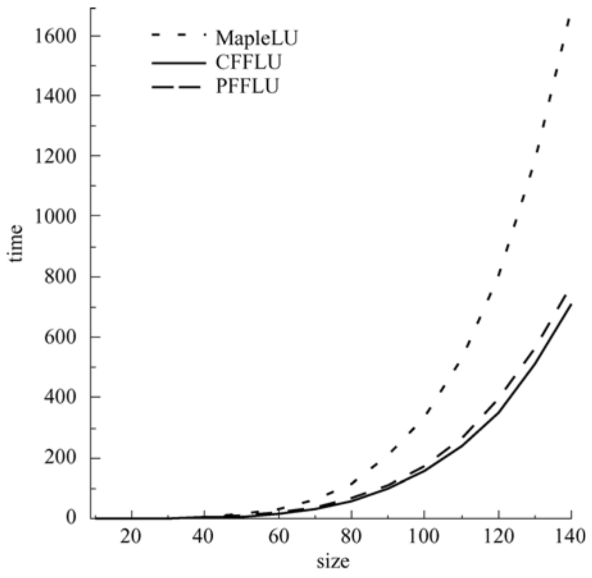

Fig. 1 Times for random integer matrices

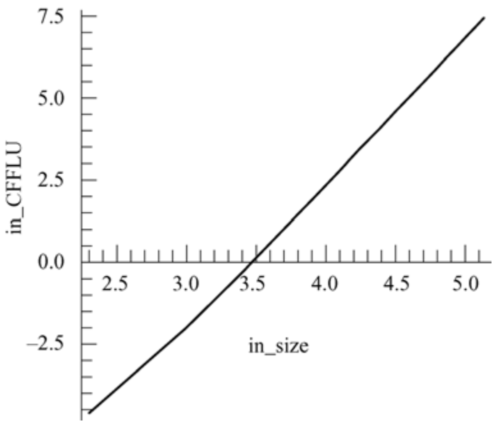

Fig. 2 Logarithms of completely fraction free LU time and integer matrix size, slope = 4.335

Table 3 Timings for completely fraction free LU factoring of random integer matrices with length ℓ, A : = LinearAlgebra [RandomMatrix](5,5, generator = $-10^{\ell} \ldots 10^{\ell}$)

ℓ	CFFLU	PFFLU	MapleLU
400	4.00E-03	6.00E-03	9.00E-03
800	7.00E-03	1.30E-02	1.80E-02
1200	1.00E-02	2.30E-02	3.10E-02
1600	1.40E-02	3.30E-02	4.80E-02
2000	2.20E-02	5.20E-02	6.90E-02
2400	2.80E-02	6.90E-02	9.30E-02
2800	3.60E-02	9.10E-02	0.121
3200	4.40E-02	0.114	0.155
3600	4.90E-02	0.137	0.19
4000	6.30E-02	0.17	0.231
4400	7.10E-02	0.195	0.272
4800	7.90E-02	0.235	0.334
5200	7.50E-02	0.249	0.37
5600	9.40E-02	0.29	0.425
6000	9.90E-02	0.324	0.485

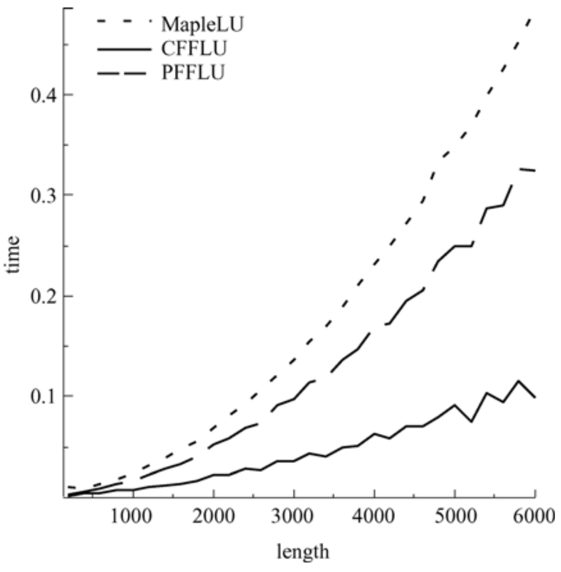

Fig. 3 Time of random integer matrices

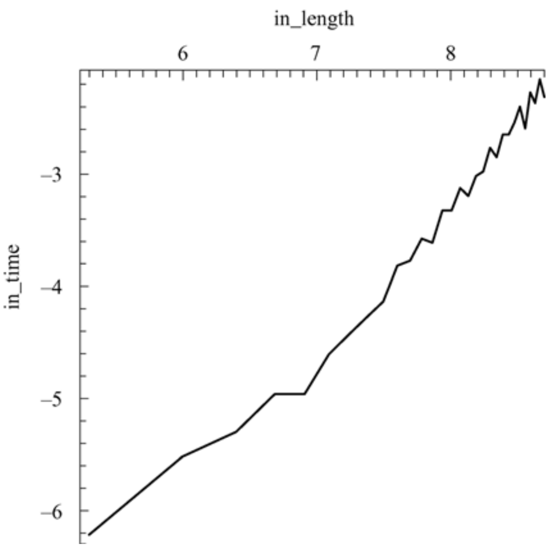

Fig. 4 Logarithms of completely fraction free LU time and integer length, slope = 1.265

the time used by completely fraction free LU factoring and the length of integer entry is $t = O(\ell^{1.265})$. This suggests (by Theorem 4) that $M(\ell) = O(\ell^{1.265})$ in the considered range.

For matrices with univariate polynomial entries, we have the following two tables. Table 4 gives Fig. 5. We can see that completely fraction free LU factoring is a little bit faster than the fraction free LU factoring given in Algorithm 2. It has a similar speed as the Maple command. It also gives the logarithms of the times used by completely fraction free LU factoring and the sizes of matrices (Fig. 6). If we use the MAPLE command Fit $(a+b\cdot t,x,y,t)$ to fit it, we find a slope equal to 5.187, i.e., we have $O(n^{2.187})$ in our implementation.

Table 4 Timings for fraction free LU factoring of random matrices with univariate entries, generated by RandomMatrix(n,n, generator = (() -> randpoly([x], degree = 4)))

n	CFFLU	PFFLU	MapleLU
5	8.00E-03	1.20E-02	2.50E-02
10	0.242	0.335	0.305
15	2.322	2.77	3.098
20	10.028	11.873	10.598
25	31.971	35.124	33.316
30	86.361	92.146	86.01
35	179.937	189.297	181.42
40	354.22	373.181	354.728
45	664.608	686.26	667.406
50	1178.374	1210.766	1179.499

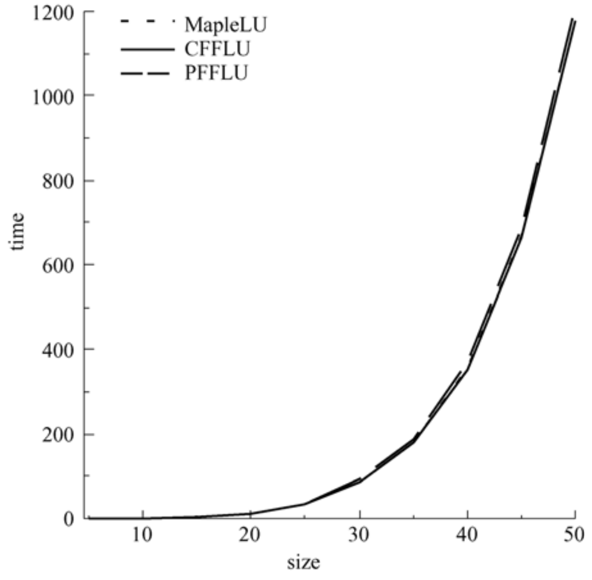

Fig. 5 Time of random matrices with univariate entries

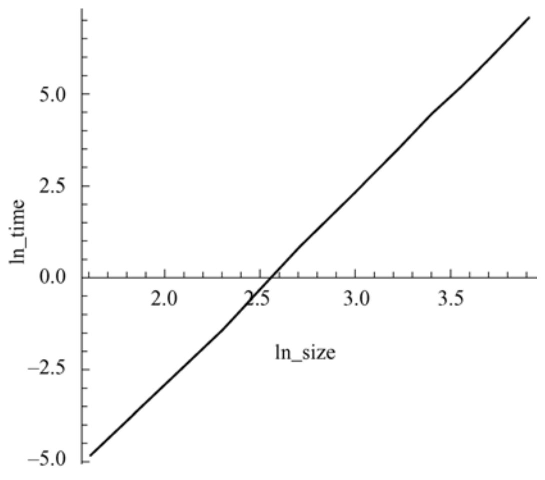

Fig. 6 Logarithms of completely fraction free LU time and random univariate polynomial matrix size, slope = 5.187

5 Application of completely fraction free LU factoring

In this section, we give applications of our completely fraction free LU factoring. Our first application is to solve a symbolic linear system of equations in a domain. We will introduce fraction-free forward and backward substitutions from Ref. [1]. Our second application is to get a new completely fraction free QR factoring, using the relation between LU factoring and QR factoring given in Ref. [22].

5.1 Fraction free forward and backward substitutions

In order to solve a linear system of equations in one domain, we need not only fraction free LU factoring of the coefficient matrix but also fraction free forward substitution (FFFS) and fraction free backward substitution (FFBS) algorithms.

Let A be a $n \times n$ matrix, and let P,L,D,U, be as in Theorem 3 with $m = n$.

Definition 3 Given a vector b in $\mathbb{I}$, fraction free forward substitution consists in finding a vector Y, such that $LD^{-1}Y = Pb$ holds.

Theorem 6 The vector Y from Definition 3 has entries in $\mathbb{I}$.

Proof From the proof of Theorem 2, if $PA = LD^{-1}U$, for $i = 1,\ldots n$, we have

$$L_{i,i} = A_{i,i}^{(i-1)},$$

and

$$D_{i,i} = A_{i-1,i-1}^{(i-2)} A_{i,i}^{(i-1)}.$$

For

$$LD^{-1}Y = Pb,$$

i.e.,

$$Y_i = \frac{D_{i,i}}{L_{i,i}}\left[b_{P_i} - \sum_{k=1}^{i-1} L_{i,k}Y_k\right],$$

where $b_{P_i} = \sum_{j=1}^{n} P_{i,j}b_j$, it is obvious that $\frac{D_{i,i}}{L_{i,i}}$ is an exact division. So the solution Y_i is fraction free for any $i = 1\ldots n$. □

Definition 4 Given the vector Y from Definition 3, fraction free backward substitution consists in finding a vector X, such that $UX = U_{n,n}Y$ holds.

Theorem 7 The vector X from Definition 4 has entries in $\mathbb{I}$, and satisfies $AX = \det(A)b$.

Proof As we have proved in Theorem 2, $U_{n,n}$ is actually the determinant of A. From the relations

$$UX = U_{n,n}Y,$$

and

$$LD^{-1}Y = Pb,$$

we get

$$PAX = LD^{-1}UX = det(A)LD^{-1}Y = Pb.$$

So

$$AX = det(A)b.$$

Let B be the adjoint matrix of A, so that $BA = \det(A)I$, where I is an identity matrix. We deduce that $X = Bb$. Since B has coefficients in $\mathbb{I}$, the result follows. □

If we have a list of linear systems with the same coefficient n by n matrix A, i.e., $Ax = b_1$, $Ax = b_2$, $Ax = b_3$, …, after completely fraction free LU decomposition of matrix $PA = LD^{-1}U$, the number of operations for solving a second linear system can be reduced from $O(n^3)$ to $O(n^2)$ for a matrix with floating point entries. Furthermore, for a matrix with symbolic entries, such as an integer matrix entry length bounded by ℓ, the time complexity can be reduced from $O(n^3M(n \log n + n\ell))$ to $O(n^2M(n \log n + n\ell))$.

Here we give our fraction free forward and backward substitution algorithms using the same notation as before.

Algorithm 3 Fraction free forward substitution

```
Input: Matrices $L, D, P$ and a vector b
Output: $A$ $n \times 1$ vector $Y$, such that $LD^{-1}Y = Pb$
For $i$ from 1 to $n$, do
    $b_{P_i} = \sum_{j=1}^{n} P_{i,j}b_j,$
    $Y_i = \frac{D_{i,i}}{L_{i,i}}\left[b_{P_i} - \sum_{k=1}^{i-1} L_{i,k}Y_k\right],$
end loops;
```

Algorithm 4 Fraction free backward substitution

```
Input: A matrix U, a vector $Y$ from $LD^{-1}Y = Pb$.
Output: A scaled solution $X$

For $i$ from $n$ by $-1$ to 1, do
    $X_i = \frac{1}{U_{i,i}}\left[U_{n,n}Y_i - \sum_{k=i+1}^{n} U_{i,k}X_k\right],$
end loops;
```

5.2 Fraction free QR factoring

In a numerical context, the QR factoring of a matrix is well-known. We recall that this accounts to find Q and R such that $A = QR$, with Q being an orthonormal matrix (i.e., having the property $Q^TQ = QQ^T = I$).

Pursell and Trimble [22] have given an algorithm for computing QR factoring using LU factoring which is simple and has some surprisingly happy side effects although it is not numerically preferable to existing algorithms. We use their idea to obtain fraction free QR factoring using left orthogonal matrices and based on our completely fraction-free LU factoring. In Theorem 8, we prove the existence of fraction free QR factoring and give an algorithm using completely fraction-free LU factoring to get fraction free QR factoring of a given matrix, i.e., how to orthogonalize a given set of vectors which are the columns of the matrix.

Theorem 8 Let $\mathbb{I}$ is an integral domain and A be a $n \times m$ matrix whose columns are linearly independent vectors in $\mathbb{I}$. There exist a $m \times m$ lower triangular matrix $L \in \mathbb{I}^n$, a $m \times m$ diagonal matrix $D \in \mathbb{I}^n$, a $m \times m$ upper triangular matrix $U \in \mathbb{I}^n$ and a $n \times m$ left orthogonal matrix $\Theta \in \mathbb{I}^n$, such that

$$\left[A^TA|A^T\right] = LD^{-1}\left[U|\Theta^T\right]. \tag{4}$$

Definition 5 In the condition of Theorem 8, the fraction free QR factoring of A is

$$A = \Theta D^{-1}R, \tag{5}$$

where $R = L^T$.

Proof The linear independence of the columns of A implies that $A^T A$ is a full rank matrix. Hence, there are $m \times m$ fraction free matrices, L, D and U as in Theorem 2, such that

$$A^TA = LD^{-1}U. \tag{6}$$

Applying the same row operations to the matrix A^T, we have a fraction free $m \times n$ matrix. Let us call this fraction free $m \times n$ matrix as Θ^T, which is

$$\Theta^T = DL^{-1}A^T, \tag{7}$$

then

$$\Theta^T\Theta = \left(DL^{-1}A^T \cdot A\left(DL^{-1}\right)^T\right)$$
$$= DL^{-1}\left(LD^{-1}U\right)\left(DL^{-1}\right)^T = U\left(DL^{-1}\right)^T.$$

Because $\Theta^T\,\Theta$ is symmetric and both U and $(DL^{-1})^T$ are upper triangular matrices, $\Theta^T\,\Theta$ must be a diagonal matrix. So the columns of Θ are left orthogonal and in $\mathbb{I}^n$ based on Theorem 6.

Based on Eq. 7, we have $A^T = LD^{-1}\Theta^T$, i.e., $A = \Theta(LD^{-1})^T = \Theta(D^T)^{-1}L^T = \Theta D^{-1}L^T$. Set $R = L^T$, then R is a fraction free upper triangular matrix. □

In the following Algorithm 5, we assume the existence of a function CFFLU implementing Algorithm 1.

Algorithm 5 Fraction free QR factoring
Input: A $n \times m$ matrix A
Output: Three matrices Θ, D, R, where Θ is a $n \times m$ left orthogonal matrix, D is a $m \times m$ diagonal matrix, R is a $m \times m$ upper triangular matrix and $A = \Theta D^{-1}R$

```
1) compute $B := [A^T A|A^T]$;
2) $L, D,$ U := CFFLU(B);
3) for $i = 1,\ldots,m$, $j = 1,\ldots,n$ $\Theta_{j,i} := U_{i,m+j}$, end loops;
4) for $i = 1,\ldots,m$, $j = 1,\ldots,m$ $R_{j,i} := L_{i,j}$, end loops. □
```

Example 3 We revisit the example in Pursell and Trimble [22].

$$a_1 := \begin{bmatrix}0\\1\\0\\1\end{bmatrix},\quad a_2 := \begin{bmatrix}-2\\3\\0\\1\end{bmatrix},\quad a_3 := \begin{bmatrix}1\\1\\1\\5\end{bmatrix}.$$

Let A be the 4×3 matrix containing a_k as its kth column. Then

$$A^TA = \begin{bmatrix}2 & 4 & 6\\4 & 14 & 6\\6 & 6 & 28\end{bmatrix}.$$

Applying the fraction free row operation matrices L and D to the augmented matrix $[A^T A|A^T]$, we have the fraction free matrix U, where

$$L = \begin{bmatrix}2 & & \\4 & 12 & \\6 & -12 & 1\end{bmatrix},\quad D = \begin{bmatrix}2 & & \\ & 24 & \\ & & 12\end{bmatrix},$$

$$U = \begin{bmatrix}2 & 4 & 6 & 0 & 1 & 0 & 1\\ & 12 & -12 & -4 & 2 & 0 & -2\\ & & 48 & -12 & -12 & 12 & 12\end{bmatrix}.$$

Notice that the matrix U differs from that given in Pursell and Trimble [22], which is as follows:

$$\left[\begin{array}{ccc|cccc}2 & 4 & 6 & 0 & 1 & 0 & 1\\ & 6 & -6 & -2 & 1 & 0 & -1\\ & & 4 & -1 & -1 & 1 & 1\end{array}\right].$$

This is because they implicitly divided each row by the GCD of the row. They also observed for this example that cross-multiplication is not needed during Gaussian elimination and the squares of the each row on the right side equal the diagonal of the left side of the matrix. However, these observations are specific to their particular numerical example. Any change to these initial vectors will invalidate their observation, as Example 4 will show.

Now from our fraction free *QR* Theorem 8, we have

$$R=L^T=\begin{bmatrix}2 & 4 & 6\\ & 12 & -12\\ & & 1\end{bmatrix},$$

$$\Theta^T=\begin{bmatrix}0 & 1 & 0 & 1\\ -4 & 2 & 0 & -2\\ -12 & -12 & 12 & 12\end{bmatrix} \text{ or}$$

$$\Theta=\begin{bmatrix}0 & -4 & -12\\ 1 & 2 & -12\\ 0 & 0 & 12\\ 1 & -2 & 12\end{bmatrix}.$$

So the fraction free QR factoring of matrix A is as follows:

$$\Theta D^{-1}R=\begin{bmatrix}0 & -4 & -12\\ 1 & 2 & -12\\ 0 & 0 & 12\\ 1 & -2 & 12\end{bmatrix}\begin{bmatrix}2 & & \\ & 24 & \\ & & 12\end{bmatrix}^{-1}\begin{bmatrix}2 & 4 & 6\\ & 12 & -12\\ & & 1\end{bmatrix}$$
$$=\begin{bmatrix}0 & -2 & 1\\ 1 & 3 & 1\\ 0 & 0 & 1\\ 1 & 1 & 5\end{bmatrix}$$
$$=A.$$

Example 4 We slightly change the vector a_1 and keep the other vectors the same as in Example 3.

$$a_1:=\begin{bmatrix}0\\2\\0\\1\end{bmatrix},\quad a_2:=\begin{bmatrix}-2\\3\\0\\1\end{bmatrix},\quad a_3:=\begin{bmatrix}1\\1\\1\\5\end{bmatrix}.$$

Let C be the 4 × 3 matrix containing a_k as its k^{th} column. Then

$$C^TC=\begin{bmatrix}5 & 7 & 7\\ 7 & 14 & 6\\ 7 & 6 & 28\end{bmatrix}.$$

Applying the fraction free row operation matrices L and D to the augmented matrix $[C^T C|C^T]$, we have the fraction free matrix U, where

$$L=\begin{bmatrix}5 & & \\ 7 & 21 & \\ 7 & -19 & 1\end{bmatrix},\quad D=\begin{bmatrix}5 & & \\ & 105 & \\ & & 21\end{bmatrix},$$

$$U=\begin{bmatrix}5 & 7 & 7 & 0 & 2 & 0 & 1\\ & 21 & -19 & -10 & 1 & 0 & -2\\ & & 310 & -17 & -34 & 21 & 68\end{bmatrix}.$$

We can verify that the square of the second row on the right side of matrix U is not equal to the diagonal of the left side of the matrix U. It means the observation of Pursell and Trimble is not valid in this example.

The fraction free QR factoring of matrix C is as follows:

$$\Theta D^{-1}R=\begin{bmatrix}0 & -10 & -17\\ 2 & 1 & -34\\ 0 & 0 & 21\\ 1 & -2 & 68\end{bmatrix}\begin{bmatrix}5 & & \\ & 105 & \\ & & 21\end{bmatrix}^{-1}\begin{bmatrix}5 & 7 & 7\\ & 21 & -19\\ & & 1\end{bmatrix}$$
$$=\begin{bmatrix}0 & -2 & 1\\ 2 & 3 & 1\\ 0 & 0 & 1\\ 1 & 1 & 5\end{bmatrix}$$
$$=C.$$

6 Conclusions

The main contributions of this paper are firstly the definition of a new fraction free format for matrix LU factoring, with a proof of the existence; then from the proof, a general way for computing the new format and complexity analysis on the two main algorithms.

One of the difficulties constantly faced by computer algebra systems is the perception of new users that numerical techniques carry over unchanged into symbolic contexts. This can force inefficiencies on the system. For example, if a system tries to satisfy user preconceptions that Gram-Schmidt factoring should still lead to QR factors with Q orthonormal, then the system must struggle with square-roots and fractions. By defining alternative forms for the factors, the system can take advantage of alternative schemes of computation.

In this paper, the application of the completely fraction free LU factoring format gives a new fraction-free QR factoring format which avoid the square-roots and fractions problems for QR factoring. On the other hand, together with the fraction-free forward and backward substitutions, fraction-free LU factoring gives one way to solve linear systems in their own input domains.

References

1. Nakos G C, Turner P R, Williams R M. Fraction-free algorithms for linear and polynomial equations. SIGSAM Bull, ACM Press, 1997, 31(3): 11–19
2. Zhou W, Carette J, Jeffrey D J, et al. Hierarchical representations with signatures for large expression management. AISC, Springer-Verlag, LNAI 4120, 2006, 254–268
3. Sasaki T, Nurao H. Efficient Gaussian elimination method for symbolic determinants and linear systems. ACM Transactions on Mathematical Software, 1982, 8(3): 277–289
4. Kirsch B J, Turner P R. Modified Gaussian elimination for adaptive beamforming using complex RNS arithmetic. NAWC-AD Tech Rep, NAWCADWAR, 1994, 941112-50
5. Kirsch B J, Turner P R. Adaptive beamforming using RNS arithmetic. In: Proceedings of ARTH. Washington DC: IEEE Computer Society, 1993, 36–43
6. Turner P R. Gauss elimination: workhorse of linear algebra. NAWC-AD Tech Rep, NAWCAD-PAX 96-194-TR, 1996
7. Bareiss E H. Sylvester's identity and multistep integer-preserving Gaussian elimination. Mathematics of Computation, 1968, 22(103): 565–578
8. Bareiss E H. Computational solutions of matrix problems over an integral domain. J. Inst. Maths Applics, 1972, 10: 68–104
9. Gentleman W M, Johnson S C. Analysis of algorithms, a case study: determinants of polynomials. In: Proceedings of 5th Annual ACM Symp on Theory of Computing. Austin: ACM Press, 1973, 135–142
10. Griss M L. An efficient sparse minor expansion algorithm. Houston: ACM, 1976, 429–434
11. McClellan M T. The exact solution of systems of linear equations with polynomial coefficients. J ACM, 1973, 20(4): 563–588
12. Smit J. The efficient calculation of symbolic determinants. In: Proceedings of SYMSAC. New York: ACM, 1976, 105–113
13. Smit J. A cancellation free algorithm, with factoring capabilities, for the efficient solution of large sparse sets of equations. In: Proceedings of ISSAC. New York: ACM, 1981, 146–154
14. Turing A M. Rounding-off errors in matrix processes. Quart. J. Mech. Appl. Math, 1948, 1: 287–308
15. Corless R M, Jeffrey D J. The Turing factorization of a rectangular matrix. SIGSAM Bull, ACM Press, 1997, 31(3): 20–30
16. Dixon J D. Exact solution of linear equations using *p*–adic expansion. Numer. Math, 1982, 137–141
17. Moenck R T, Carter J H. Approximate algorithms to derive exact solutions to systems of linear equations. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation. Berlin: Springer-Verlag, 1979, 65–73
18. Storjohann A. High-order lifting and integrality certification. Journal of Symbolic Computation, 2003, 36(3–4): 613–648
19. Storjohann A. The shifted number system for fast linear algebra on integer matrices. Journal of Complexity, 2005, 21(4): 609–650
20. von zur Gathen J, Gerhard J. Modern computer algebra. London: Cambridge University Press, 1999
21. Krick T, Pardo L M, Sombra, M. Sharp estimates for the arithmetic Nullstellensatz. Duke Mathematical Journal, 2001, 109(3): 521–598
22. Pursell L, Trimble S Y. Gram-Schmidt orthogonalization by Gaussian elimination. American Math. Monthly, 1991, 98(6): 544–549