RESEARCH ARTICLE

Wenqin ZHOU (✉), David J. JEFFREY

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

>LUDecomposition(A, method = FractionFree);

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

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

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 $A\in {}^{n\times m}$ may be written

$$F_1PA = LF_2U, \qquad (2)$$

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

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 $\mathbb{D}$ be an integral domain and $A = [a_{i,j}]_{i\leqslant n, j\leqslant m}$ be a matrix in $\mathbb{D}^{n\times m}$ 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.

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

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

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

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

Let us define $A_{i,j}^{(k)}$ by $A_{i,j}^{(k)}=\begin{matrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} & a_{1,j} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} & a_{2,j} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k} & a_{k,j} \\ a_{i,1} & a_{i,2} & \cdots & a_{i,k} & a_{i,j} \end{matrix}\;.$
Hence, these are in 𝕀.

. j j

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

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

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 𝔻 and $n \leqslant m$. On input A, Algorithm 1 outputs four matrices P,L,D,U with entries in 𝔻 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

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

$$\frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A\text{A}}{}$$

and

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

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 $\deg\left(A^{(k)}_{i,j}\right)\leqslant kd$.

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

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

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

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 $A\in$, i.e. we only consider the growth of length during the factoring. In case 3, we will analyze the cost of both algorithms with $A\in\ [x]$. For more cases, such as $A\in\ [x_1,\ldots,x_m]$, 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 $O(n^3M(nd))$ operations in $\mathbb{K}$.

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

For a matrix $A=[a_{i,j}]_{n\times n}$ with univariate polynomial entries in $[x]$, 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 $O\left(n^3M\left(n^2d\ell+nd^2\right)\right)$ 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 $\ell=\max_{i,j}\lambda\left(a_{i,j}\right)$ and case 3 be the case $a_{i,j}\in\ [x]$ with $d=\max_{i,j}\deg(a_{i,j})+1$ and $\ell=\max_{i,j}\lambda\left(a_{i,j}\right)$. From Lemma 2, at each step k, $\deg\left(A^{(k)}_{i,j}\right)\leqslant kd$ in case 1 and $\lambda\left(A^{(k)}_{i,j}\right)\leqslant k(\ell+\log k)$ in case 2, and λ

$\log\log(nd(\ell+d))+ndM(n(\ell+\log n+d)\log(nd(\ell+d)))$ $\log(n(\ell+d))$.

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 $a_{i,j}\in$ with $\ell=\max_{i,j}^{\lambda}\ a_{i,j}$ and case 3 is $a_{i,j}\in\ [x]$ 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^{th}, deg $A_{i,j}^{(k)}) \leqslant kd$ in case 1, $\lambda\ A_{i+\text{f}kd=}^{(k))}\ _{i}$ ⋠⩽⩽⩽⩽⩽⩽⩽⩽⩽⩽⩽⩽⩽⩽⩾⩽⩽

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

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

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

where $b_{P_i} = \quad \overset{n}{\underset{j=}{}}$

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. □

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

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

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:

ΘD^{-}