

# New Forms for the LU Factoring of Matrices

David Jeffrey

Applied Mathematics and ORCCA,  
University of Western Ontario

ACA 2008, Hagenberg



## The Starting Point

In standard, particularly *numerical*, linear algebra, we learn to factor a matrix  $A$  as  $PA=LU$ , for example, using `MATLAB`

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 5 & 10 & 15 \\ -1 & -6 & -19 \\ 6 & 5 & 15 \end{pmatrix} =$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0.8333 & 1 & 0 \\ -0.1667 & -0.8857 & 1 \end{pmatrix} \begin{pmatrix} 6 & 5 & 15 \\ 0 & 5.8333 & 2.5000 \\ 0 & 0 & -14.2857 \end{pmatrix}$$



## Applications of LU

- Turing used it for theoretical analysis of Gaussian elimination
- Given a sequence of equations to solve,

$$Ax_1 = b_1$$

$$Ax_2 = b_2$$

$$\vdots \quad \vdots \quad \vdots$$

after  $O(n^3)$  operations to factor  $A$ , each solution is  $O(n^2)$ .

- Iterative calculations, such as inverse power iteration are applications.



## Extension by MATLAB

Early MATLAB and other programs returned an error if the matrix  $A$  were not square or not invertible.

Recent versions of MATLAB and MAPLE return results for non-square matrices. However, they disagree. MATLAB says:

$$\begin{pmatrix} 5 & 10 & 15 & 20 \\ -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 5 & 6 & -1 & -12 \\ 4 & 9 & 16 & 29 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -0.2 & 1 & 0 & 0 \\ 0.2 & -0.75 & 1 & 0 \\ 1.0 & 1.0 & 0 & 1 \\ 0.8 & -0.25 & 0 & -0.5 \end{pmatrix} \begin{pmatrix} 5 & 10 & 15 & 20 \\ 0 & -4 & -16 & -12 \\ 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & -20 \end{pmatrix}$$



## Extension by MAPLE

$$\begin{pmatrix} 5 & 10 & 15 & 20 \\ -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 5 & 6 & -1 & -12 \\ 4 & 9 & 16 & 29 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1/5 & 1 & 0 & 0 & 0 \\ 1/5 & -3/4 & 1 & 0 & 0 \\ 1 & 1 & -10/3 & 1 & 0 \\ 4/5 & -1/4 & 5/3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 & 10 & 15 & 20 \\ 0 & -4 & -16 & -12 \\ 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

## Full Rank Factors

Recall facts about rank of a matrix.

- Column rank = Row rank
- Rank  $\leq$  Columns, and less than or equal to rows.
- Rank equals non-zero rows in row-reduced form.
- Numerically, rank is difficult to compute.

Given a matrix  $A$  that is  $m \times n$  having rank  $r$ , define  $F$  and  $G$  as  $m \times r$  and  $r \times n$ , such that

$$A = FG .$$

Then this is a full rank factoring.



## Full-rank LU factors

Since the rank equals the non-zero rows in a (properly computed) LU reduction, we only have to drop the zero rows.

$$\begin{pmatrix} 5 & 10 & 15 & 20 \\ -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 5 & 6 & -1 & -12 \\ 4 & 9 & 16 & 29 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1/5 & 1 & 0 & 0 & 0 \\ 1/5 & -3/4 & 1 & 0 & 0 \\ 1 & 1 & -10/3 & 1 & 0 \\ 4/5 & -1/4 & 5/3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 & 10 & 15 & 20 \\ 0 & -4 & -16 & -12 \\ 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



## FRLU with permutations

Since we shall be using a computer to obtain the FRLU factors, let us require them in their most useful form. We use permutation matrices to re-arrange the rows and columns. We shall write

$$A = P_r L U P_c .$$

Here  $P_r$  permutes rows and  $P_c$  permutes columns. For example

$$\begin{pmatrix} 0 & 0 & 14 \\ 6 & 9 & 15 \\ 2 & 3 & -2 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1/3 & -1/2 \end{pmatrix} \begin{pmatrix} 6 & 15 & 9 \\ 0 & 14 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

## In Block Notation

$$A = P_r \begin{pmatrix} \mathcal{L} \\ \mathcal{M} \end{pmatrix} (\mathcal{U} \quad \mathcal{V}) P_c .$$

where

$$\begin{aligned} A &= m \times n \\ P_r &= m \times m \\ \mathcal{L} &= r \times r \\ \mathcal{M} &= (m - r) \times r \\ \mathcal{U} &= r \times r \\ \mathcal{V} &= r \times (n - r) \\ P_c &= n \times n \end{aligned}$$

and  $\mathcal{L}$  and  $\mathcal{U}$  are invertible.

## Overdetermined systems in Engineering

I always tell my engineering students that *engineering* linear algebra is different from *mathematical* linear algebra. Suppose we have a problem with  $n$  unknowns.

- Mathematicians need  $n$  equations because
  1. Mathematicians are perfect and never make mistakes.
  2. Each of the  $n$  equations will be independent.
  3. Every coefficient in the  $n$  equations will be correct.
- Engineers make mistakes and need more than  $n$  equations, which they check for consistency to check for errors.

Suppose an engineer has  $m$  equations in  $n$  unknowns, and derives a set of equations  $Ax = b$  for them. Then

$$Ax = P_r L U P_c x = b$$

Now  $P_c x = [x_b \ x_f]^T$  where  $x_b$  are the bound variables and  $x_f$  are the free variables. Also  $P_r^T b = [b_b \ b_c]^T$ . Then the bound variables are determined by

$$x_b = U^{-1}(\mathcal{L}^{-1}b_b - \mathcal{V}x_f).$$

The consistency condition is

$$\mathcal{M}\mathcal{L}^{-1}b_b = b_c$$

## Moore—Penrose by FRLU

Let  $A^+$  be the Moore—Penrose Inverse. It is defined by the equations  $AA^+A = A$ ,  $A^+AA^+ = A^+$ ,  $(AA^+)^H = AA^+$ ,  $(A^+A)^H = A^+A$ . If  $Ax = b$  is an inconsistent system, then  $x = A^+b$  is the least-squares solution, namely  $\|Ax - b\|$  is minimum and  $\|x\|$  is minimum.

Surprisingly, it is given by a formula

$A^+ = G^T(GG^T)^{-1}(F^TF)^{-1}F^T$ . For our FRLU, this becomes

$$A^+ = [P_r L (U P_c A^T P_r L)^{-1} U P_c]^T$$

Notice the cute way it mirrors the decomposition



But wait, there is more!

We can combine FRLU with fraction-free methods, to obtain  
FFFRLU !

## Troubles with LU: why fraction-free?

The form of LU factors is dominated by floating-point usage:

$$PA = \begin{pmatrix} 1 & 0 & 0 & \dots \\ \ell_{21} & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots \\ 0 & u_{22} & \dots \\ \vdots & \vdots & \vdots \end{pmatrix}$$

If the entries of  $A$  and the arithmetic are floating-point, then the form makes sense, but suppose the arithmetic is exact and the entries of  $A$  come from some integral domain. Forcing the diagonal entries to 1 means that all the other entries are quotients.

Can we devise a format in which we can avoid quotients?

## One attempt: Maple “FractionFree”

We shall start with matrices over  $\mathbb{Z}$  and end with an example over  $\mathbb{Z}[x]$ . Maple command

`LUdecomposition(A,method=FractionFree).`

$$\begin{aligned}
 A &= \begin{pmatrix} -69 & -59 & 34 \\ 1 & 71 & 55 \\ 57 & -65 & 13 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{69} & 1 & 0 \\ -\frac{19}{23} & -\frac{981}{605} & 1 \end{pmatrix} \begin{pmatrix} -69 & -59 & 34 \\ 0 & \frac{4840}{69} & \frac{3829}{69} \\ 0 & 0 & \frac{79296}{605} \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{69} & -\frac{1}{69} & 0 \\ -\frac{19}{23} & -\frac{327}{13915} & -\frac{1}{4840} \end{pmatrix} \begin{pmatrix} -69 & -59 & 34 \\ 0 & -4840 & -3829 \\ 0 & 0 & -634368 \end{pmatrix}
 \end{aligned}$$

## Second attempt: Corless & Jeffrey

Recall pseudo-division of polynomials  $A(x)$  divided by  $B(x)$ . We have  $A, B \in \mathbb{Z}[x]$  and want  $Q, R \in \mathbb{Z}[x]$  also.

$$\beta A(x) = Q(x)B(x) + R(x) .$$

The  $\beta$  factor inflates the  $A(x)$  polynomial so that the subsequent division does not introduce fractions. We tried the same trick with matrices:

$$F_1 A = L F_2 U .$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & -21 & 0 \\ 0 & 0 & 0 & -420 \end{bmatrix} \begin{bmatrix} 3 & 4 & -2 & 1 & -2 \\ 1 & -1 & 2 & 2 & 7 \\ 4 & -3 & 4 & -3 & 2 \\ -1 & 1 & 6 & -1 & 1 \end{bmatrix} = \\
 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -28 & -25 & 1 & 0 \\ 140 & 140 & -56 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -21 \end{bmatrix} \\
 \begin{bmatrix} 3 & 4 & -2 & 1 & -2 \\ 0 & -7 & 8 & 5 & 23 \\ 0 & 0 & 20 & 72 & 159 \\ 0 & 0 & 0 & -556 & -1112 \end{bmatrix}$$

## Third attempt: New fraction-free form

Completely fraction free LU in a concise form. If the entries of  $A$  belong to some domain, then a computer program returns 3 matrices  $L$ ,  $D$ ,  $U$  with the understanding that they are combined as follows.

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

Apply this to previous example.

$$P \begin{pmatrix} 1 & 0 & 0 & 0 \\ 4 & 1 & 0 & 0 \\ -1 & 0 & 2 & 0 \\ 3 & 7 & 5 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix}^{-1} \begin{pmatrix} 1 & -1 & 2 & 2 & 7 \\ 0 & 1 & -4 & -11 & -26 \\ 0 & 0 & 8 & 1 & 8 \\ 0 & 0 & 0 & 556 & 1112 \end{pmatrix}$$

## Basic structure

Suppose we are computing  $LU$  by Gaussian elimination. Then there will be pivots  $p_1, p_2, \dots$ , then

$$\begin{pmatrix} p_1 & 0 & 0 & \dots \\ l_{21} & p_2 & 0 & \dots \\ l_{31} & l_{32} & p_3 & \dots \\ \vdots & & & \end{pmatrix} \begin{pmatrix} p_1 & 0 & 0 & \dots \\ 0 & p_1 p_2 & 0 & \dots \\ 0 & 0 & p_2 p_3 & \dots \\ \vdots & & & \end{pmatrix}^{-1} \begin{pmatrix} p_1 & u_{12} & u_{13} & \dots \\ 0 & p_2 & u_{23} & \dots \\ 0 & 0 & p_3 & \dots \\ \vdots & & & \end{pmatrix}$$

Notice that since the  $D$  matrix is completely predictable, it does not need to be stored. We can return to the trick of numerical linear algebra of storing the whole output in the  $A$  matrix.

## Combine with Full Rank

Thus, given a matrix  $A$  as above, we factor it

$$A = PLD^{-1}U,$$

with the matrix dimensions being

$$P \quad m \times m$$

$$L \quad m \times r$$

$$D \quad r \times r$$

$$U \quad r \times n$$

When the entries of  $A$  lie in some integral domain  $\mathbb{K}$ , for example, the integers  $\mathbb{Z}$  or polynomials  $\mathbb{Z}[x]$ , then the other matrices are also from that domain.

## Implementation notes

1. We make the factors unique by
  - Removing GCDs from the rows of  $U$ , then the columns of  $L$ .
  - For integers, we make the leading entries in  $U$  positive.
2. The computation can be performed using only exact divisions using fraction-free Gaussian elimination (Bareiss).
3. To accommodate different pivoting strategies, the Maple function takes as an argument a pivoting selector, which is a function that takes a column as input and returns the desired pivot.
  - FirstNonZero.
  - SmallestNonZero.
  - Largest (Partial pivoting from floating-point computation).

## Exact and non-exact divisions

An exact division costs less than a non-exact division:

- **exact division** ( $\frac{20}{5} = 4$ ): exact division **once**
- **non-exact division** ( $\frac{46}{10}$ ):
  1. find gcd for denominator and numerator; (gcd computation **once**)  
 $\text{gcd}(46, 10) = 2$ ;
  2. divide denominator and numerator by their gcd. (exact division **twice**)  

$$\frac{46}{10} = \frac{46/2}{10/2} = \frac{23}{5}$$

**Comment:** completely fraction free LU factoring avoids GCD computation.

## Completely Fraction Free LU Factoring

### Theorem

Let  $\mathbb{I}$  be an integral domain and  $A$  be a full-rank matrix in  $\mathbb{I}^{n \times m}$  with  $n \leq m$ . Then,  $A$  may be written  $A = P^{-1}LD^{-1}U$ , where

- $P$  is a permutation matrix,
- $L$  is lower triangular, with  $L_{i,k} = A_{i,k}^{(k-1)}$  ( $n \geq i \geq k \geq 1$ ),
- $U$  is upper triangular, with  $U_{k,j} = A_{k,j}^{(k-1)}$  ( $m \geq j \geq k \geq 1$ ),
- $D$  is diagonal, with  $D_{k,k} = A_{k-1,k-1}^{(k-2)} A_{k,k}^{(k-1)}$  ( $n \geq k \geq 1$ ),

$$\text{with } A_{i,j}^{(k)} = \begin{vmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} & a_{1,j} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} & a_{2,j} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k} & a_{k,j} \\ a_{i,1} & a_{i,2} & \cdots & a_{i,k} & a_{i,j} \end{vmatrix}.$$

In particular, all matrices have entries in  $\mathbb{I}$ .



## Complexity of Fraction Free LU Factoring

Only for integer matrices; for matrices in  $\mathbb{K}[x]$  or  $\mathbb{Z}[x]$ , refer to my thesis.

Algorithm: fraction free LU with **exact divisions** only based on Theorem 1.

### Theorem

Let  $A = [a_{i,j}]_{n \times n} \in \mathbb{Z}$  and  $\text{length}(a_{i,j}) \leq \ell$ . The time complexity of completely fraction free LU factoring for  $A$  is in  $O(n^3 M(n \log n + n\ell))$  word operations, where  $M(\ell)$  is the cost of multiplying two integers with length  $\ell$ .

Remark: The time complexity of partially fraction free LU factoring for  $A$  will be in  $O((n^2 \log(n \log n + n\ell) + n^3)M(n \log n + n\ell))$  because of **GCD computations**.

Remark: Special cases.

	PFFLU	CFFLU
Case 1: $\ell$ fixed, $n$ changes	$O(n^3 M(n \log n))$	$O(n^3 M(n \log n))$

# Benchmarks on Integer matrices

**Figure:** Time versus length and size.

**Figure:** Time versus length.

## Application to linear system solving

For instance,  $Ax = b_1, Ax = b_2, Ax = b_3, \dots,$

- keep all the computations in one domain.

Solve  $Ax = b$  in one domain:

1. Completely Fraction Free LU ( $PA = LD^{-1}U$ ),
2. Fraction Free Forward Substitution ( $LD^{-1}Y = Pb$ ), The calculation  $Y = DL^{-1}Pb$  can be done fraction-free and  $Y$  is fraction free.
3. Fraction Free Backward Substitution ( $UX = U_{n,n}Y$ ),
4. Fraction free solution  $X$  with  $AX = \det(A)b$ .



- floating point entries:  $O(n^3)$  to  $O(\quad)$
- integer entries (length  $\ell$ ),  $O(n^3M(n \log n + n\ell))$  to  $O(\quad M(n \log n + n\ell))$ .

## Application to linear system solving

For instance,  $Ax = b_1, Ax = b_2, Ax = b_3, \dots,$

- keep all the computations in one domain.  
Solve  $Ax = b$  in one domain:
  1. Completely Fraction Free LU ( $PA = LD^{-1}U$ ),
  2. Fraction Free Forward Substitution ( $LD^{-1}Y = Pb$ ),
  3. Fraction Free Backward Substitution ( $UX = U_{n,n}Y$ ),
  4. Fraction free solution  $X$  with  $AX = \det(A)b$ .
- After  $PA = LD^{-1}U$ , **cost for solving equations reduced:**
  - floating point entries:  $O(n^3)$  to  $O(n^2)$
  - integer entries (length  $\ell$ ),  $O(n^3M(n \log n + n\ell))$  to  $O(n^2M(n \log n + n\ell))$ .

## Application to Moore—Penrose Inverse

We have that if  $A = FG$  then  $A^+ = G^T(GG^T)^{-1}(F^T F)^{-1}F^T$ . For our decomposition, we have  $F = P_r L$  and  $G = D^{-1}UP_c$ . Substituting and simplifying, we obtain

$$A^+ = [P_r L (UP_c A^T P_r L)^{-1} UP_c]^T$$

Which is independent of  $D$  and  $D^{-1}$ . Thus the Moore-Penrose inverse can be computed by forming  $UP_c A^T P_r L$  as a fraction-free product, followed by a fraction-free inverse and then more fraction-free products.