LU Factoring of Non-Invertible Matrices

D. J. Jeffrey
Department of Applied Mathematics,
The University of Western Ontario,
London, Ontario, Canada N6A 5B7

Revised submission to Communications in Computer Algebra, 2010

Abstract

The definition of the LU factoring of a matrix usually requires that the matrix be invertible. Current software systems have extended the definition to non-square and rank-deficient matrices, but each has chosen a di erent extension. Two new extensions, both of which could serve as useful standards, are proposed here: the first combines LU factoring with full-rank factoring, and the second extension combines full-rank factoring with fraction-free methods. Amongst other applications, the extension to full-rank, fraction-free factoring is the basis for a fraction-free computation of the Moore | Penrose inverse.

1 Introduction

Mathematical software systems occasionally take the initiative away from mainstream mathematics and create extensions of mathematical theory. The example of interest here concerns the LU factoring of matrices. Many textbooks restrict their definitions to square invertible matrices, and early versions of Maple, Mathematica and Matlab followed the textbooks by implementing LU-factoring routines that gave error messages for non-square matrices, and also gave error messages if the square matrix were singular.

More recent versions of these software systems have been leading the way in extending the definition of LU factoring, however, they have been leading `madly o in all directions' [18]. Recent versions of Matlab and Maple will now return results for all matrices, but not the same results. For example, two sets of LU factors for the same matrix are given below; the first line shows the factors returned by Matlab 7.9 and the second shows those returned by Maple 13.

$$\begin{array}{cccc} 5 & 10 & 15 & 20 \\ -1 & -6 & -19 & -16 \end{array}$$

both were interested in least-squares corrections used in surveying, and it was only later that Crout [9] included non-symmetric equations in his treatment. Turing wrote an influential paper [23] stating LU factoring in its modern matrix notation, and explicitly connecting it to Gaussian elimination. Dwyer [12] also credits Banachiewicz[3] with anticipating matrix factoring ideas[2].

The first part of this paper defines a form for LU factors that is more useful than the variations at present o ered by software systems, or books. The second part takes up fraction-free algorithms developed for linear system solving [14, 4]. There have been various attempts to apply the fraction-free idea to LU factoring [7, 19, 24]. Here we follow [24] and define a combined full-rank and fraction-free form.

2 Full-rank factoring and LU

Given a rectangular matrix A that is × and has rank , a full rank factoring of A consists of two matrices F and G having dimensions × and × such that $A = FG$. An introduction to the full-rank factoring of a matrix can be found in [21]. Clearly, full-rank factoring is not unique.

In the context of exact computation, an obvious way to find the rank of a matrix is to use Gaussian elimination to reduce the matrix to row-echelon form and then to count the nonzero rows. Numerical linear algebraists will immediately point out that rank is di cult to calculate using approximate arithmetic, and that methods other than Gaussian elimination are preferred for the estimation of rank. This is an important point for implementation, but the general proposal here can be presented using Gaussian elimination as the basis. Since Gaussian elimination is equivalent to L factoring [23], it is natural to extend L factoring to non-square or rank-deficient matrices by using Gaussian elimination to obtain lower- and upper-triangular matrices L and , and then to discard all zero rows in and corresponding columns of L. For example, using MAPLE's L factors given in (2), we discard the elements shown in bold to obtain a full-rank LU factoring.

One characteristic of a rank-deficient matrix is the possibility that in the factor an echelon form will replace the purely triangular form of an invertible matrix, as can be seen in (2). By reordering the columns of we can recover the more convenient form, and if a computer is performing the factoring, we can reasonably request the software to complete this small extra task for us. This suggests the following theorem, which is stated in a form that would be equally suitable for exact computation or approximate computation, provided again, we note the di culty of computing rank numerically.

r 1 *G e a a A, d e × a d a , e e e a fac*

$$A = {}_r L \quad {}_c , \qquad (3)$$

, e e $_r$ a × e a a , L a × , e a a a , a × e a a a a d $_c$ a ×

pre581(the)-31(or5)-508(matrix)]TJ/F81 9.963 Tf 49.132 9.963 Td[(P)]TJ/F18 6.974 Td[(r6.39 -1.49 Td[(r)]TJ /

2.2 Application to analyzing rectangular systems

An application of the factoring that exists in any first course on matrix theory is the standard topic is deciding how many solutions there are to a given system of equations. Most books begin by listing three possibilities [2], namely, a system can have no solution, one solution or an infinite number of solutions; after that, they treat particular examples by reducing an augmented matrix to row-echelon form, and then apply an ad c analysis. With the new LU factors, the analysis is quick. Suppose there are equations in unknowns in the usual form A = , with A having rank . We obtain the full-rank LU factors:

$$A = {}_r L \quad {}_c = .$$

We first separate the bound and free variables, by writing ${}_c = [\ {}_b \ \ {}_f]^T$, with ${}_b$ being the bound variables and ${}_f$ the − free variables. We also separate the right-hand side into corresponding constants: ${}_r{}^{1} = [\ {}_b \ \ {}_c]^T$. Now we can decide whether solutions exist by checking the consistency condition,

$$M L^{1} {}_b = {}_c . \tag{4}$$

If this equation is satisfied, then the system is consistent, and we can write the bound variables in terms of the free variables as

$${}_b = (LU)^{1} {}_b - U^{1} {}_f .$$

Ar508(rectangular)-7[(matrix)]TJ/F119.963Tf58.540Td[An

The iterative procedure is more clearly seen if we write $D_1 = I$ and then (8) becomes

$$D_2{}^{1}E_2D_1{}^{1}E_1A \quad = \quad A^{(3)} \ . \tag{9}$$

In words, step of a fraction-free Gaussian elimination consists of a pivoting step (not shown), a cross multiplication step and a division by the pivot from step $- 1$.

The connection with

[16] Nicholas J. Higham. *Acc ac a d S ab f N e ca A* . Society for Industrial & Applied Mathematics, 1996.

[17] Leslie Hogben, editor. *Ha db f ea a eb a*. Chapman and Hall/CRC, 2007.

[18] Stephen B. Leacock. *N e e N e* , chapter Gertrude the Governess. Wildside Press, 1911.

[19] George C. Nakos, Peter R. Turner, and Robert M. Williams. Fraction-free algorithms for linear and polynomial equations. *SIGSAM B* ., 31(3):11{19, 1997.

[20] R. Piziak and P. L. Odell. Full rank factorization of matrices. *Ma e a c Ma a e*, 72(3):193{201, 1999.

[21] Robert Piziak and P.L. Odell. *Ma T e : F Ge e a ed I e e J da F* . Chapman & Hall/CRC, 2007.

[22] Lloyd N. Trefethen and David Bau III. *N e ca L ea A eb a*. Society for Industrial & Applied Mathematics, 1997.

[23] Alan M. Turing. Rounding-o errors in matrix processes. *Q a . J. Mec . A . Ma* ., 1:287 { 308, 1948.

[24] Wenqin Zhou and D.J. Je rey. Fraction-free matrix factors: new forms for LU and QR factors. *F e f C e Sc e ce C a*, 2(1):67{80, 2008.