Sparse Exact Factorization Update

Jinhao Chen, Timothy A. Davis
Computer Science and Engineering
Texas A&M University
College Station, USA
jinhchen@tamu.edu, davis@tamu.edu

Christopher Lourenco
Mathematics
United States Naval Academy
Annapolis, USA
lourenco@usna.edu

Erick Moreno-Centeno
Industrial and Systems Engineering
Texas A&M University
College Station, USA
emc@tamu.edu

***Abstract*—To meet the growing need for extended or exact precision solvers, an efficient framework based on Integer-Preserving Gaussian Elimination (IPGE) has been recently developed which includes dense/sparse LU/Cholesky factorizations and dense LU/Cholesky factorization updates for column and/or row replacement. In this paper, we discuss our on-going work developing the sparse LU/Cholesky column/row-replacement update and the sparse rank-1 update/downdate. We first present some basic background for the exact factorization framework based on IPGE. Then we give our proposed algorithms along with some implementation and data-structure details. Finally, we provide some experimental results showcasing the performance of our update algorithms. Specifically, we show that updating these exact factorizations can be typically 10x to 100x faster than (re-)factorizing the matrices from scratch.**

***Index Terms*—sparse factorization updates, exact precision, integer-preserving Gaussian elimination**

I. Introduction

Factorization of the coefficient matrix A is critical for solving the linear system $A\mathbf{x} = \mathbf{b}$, as it improves both the efficiency and numerical stability over the direct use of A^{-1} [1]. Traditionally, this factorization is obtained via floating-point computations, since such computations can be performed quickly on computers and, even though they have limited precision, it is adequate for most applications. Nevertheless, there are applications where extended precision or exact precision is needed, such as studies of biological processes, dynamical systems and mathematical physics [2], [3], scattered data interpolation, and solving partial differential equations [4]. A natural method to obtain the exact factorization is to perform computations with full-precision rational arithmetic, which has been applied for basic solution validation in some state-of-the-art exact mathematical programming solvers: QSopt_ex [5] and SCIP:SoPlex-exact [6], [7]. Yet, these full-precision rational-arithmetic computations involve recurring greatest common divisor calculations and irregular memory usage/access; indeed, they are so expensive in both computational time and memory that they have been shown to be the bottleneck of exact linear programming solvers [8].

To address these drawbacks, an exact (dense) factorization framework based on Integer-Preserving Gaussian Elimination (IPGE), referred to as the Roundoff-Error-Free (REF) LU/Cholesky factorization, was first proposed in [9]. Under this framework, all operations are guaranteed to be integer (notably because all the therein divisions are guaranteed to have zero remainder) and the bit-length of each entry is bounded polynomially; thereby addressing the time and memory cost drawbacks of rational factorization. Specifically, a dense LU/Cholesky factorization is an order of magnitude faster than the exact rational-arithmetic factorization [10]. These results were extended in [11] where the sparse left-looking integer-preserving (SLIP) LU was derived, which was shown to exactly solve the sparse linear system $A\mathbf{x} = \mathbf{b}$ in time proportional to arithmetic work—to date the only exact factorization for unsymmetric linear systems with this asymptotically efficient complexity bound. Finally, this sparse factorization was shown to be significantly faster than sparse rational-arithmetic factorization (like those used in QSopt/Soplex).

When a factorization of the coefficient matrix A is available and a closely related matrix $\bar{A}$ is obtained by making elementary changes to A, then the factorization of $\bar{A}$ can be efficiently obtained by modifying/updating the available factorization of A. This process, known as factorization update, is fundamental for many applications [12]–[15]. The most frequently used updates are the column and/or row replacement, and the rank-1 update/downdate. The LU factorization update for change of basis (column replacement) is heavily used in the simplex algorithm [1], [13], [16], and the Cholesky factorization rank-1 update/downdate is used in the linear programming dual active set algorithm [17]. Directly applying these traditional factorization update algorithms to the REF LU/Cholesky factorization was shown to be inadequate due to bit-length growth; indeed the update can be as costly as computing a new exact factorization from scratch [18]. Therefore, a new (dense) REF factorization update, referred to as the push-and-swap approach, was derived and shown to be efficient [18]. The key idea of the push-and-swap approach is that the to-be-replaced column, say column k, is pushed out of the matrix by swapping it sequentially with the column to its right (column $k+1$). These adjacent column swaps are done so that the factorization obtained via updates is identical to factorizing the permuted matrix from scratch. Analogously, in a yet-to-be-published manuscript, Escobedo and Gudivada derive an exact (dense) rank-1 update/downdate algorithm for the exact LU/Cholesky factorization. Since most real-world

The work is partially supported by the National Science Foundation under Grant No OAC-1835499. Additionally, the third author was also partially supported by the United States Naval Academy Junior NARC.

linear programming matrices are sparse, it is critical that the REF updates exploit sparsity. That is the topic of this paper.

The paper is organized as follows. Section 2 reviews the REF factorization framework [9] and the different types of sparse exact factorization update algorithms. Section 3 presents the implementation details of the sparse REF update algorithms. Section 4 tests the efficiency of the algorithms on a set of real-world matrices from the SuiteSparse Matrix collection [19]. Finally, Section 5 draws conclusions and discusses the outlook for future research on this topic.

II. Background and Fundamentals

For the sake of simplicity, this paper denotes the integer sequence $\{\underline{k}, \underline{k}+1, \cdots, \overline{k}-1, \overline{k}\}$ as $[\underline{k}..\overline{k}]$ and makes the following assumptions: (i) $A = \{a_{i,j}\} \in \mathbb{Z}^{n\times n},\ i \in [1..n],\ j \in [1..n]$ is a nonsingular matrix; (ii) A is properly permuted such that $a_{k,k}$ is the k-th pivot; and (iii) A is factorized via REF LU as $A = LDU$. Note that these assumptions are a matter of convenience and assumption (ii) can be removed with simple modifications to the presented algorithms [18]. In this section, we first review integer-preserving Gaussian elimination (IPGE) [20]–[22], which is the basis of the REF factorization framework. Next, we discuss the theorems for sparse exact factorization update. The proofs of these theorems have been carefully derived and verified, and will appear in a future publication. Due to space limitations and the relative simplicity of other update algorithms, only the detailed algorithm for sparse exact LU update for column replacement is given.

A. Integer-Preserving Gaussian Elimination (IPGE)

IPGE is an elimination process for solving the linear system $A\mathbf{x} = \mathbf{b}$, which is defined as follows [20]–[22]:

Definition 1: For $k \in [0..n]$, let $A^k = \{a^k_{i,j}\}, i, j \in [1..n]$ denote the k-th iteration IPGE matrix of A. Then, the pivot element selected from A^{k-1} to perform the k-th IPGE iteration can be denoted as $\rho^k := a^{k-1}_{k,k}$, where $A^0 := A$ and $\rho^0 := 1$. Then, for $k \in [1..n]$, $a^k_{i,j}$ is iteratively computed as follows:

$$a^k_{i,j} = \begin{cases} a^{k-1}_{i,j} & \text{if } i = k, \\ \frac{\rho^k a^{k-1}_{i,j} - a^{k-1}_{k,j} a^{k-1}_{i,k}}{\rho^{k-1}} & \text{otherwise.} \end{cases} \tag{1}$$

The key property of (1), termed *IPGE Update*, is that the division is guaranteed to have zero remainder, and thus for all $k \in [1..n]$, all entries of A^k remain in the integer domain [20]–[22]. Therefore, exact precision is achieved without using rational arithmetic. Furthermore, as the next theorem states, each entry of A^k is indeed a subdeterminant of A [20], [21]:

Theorem 2: Let $(A)^{1,\cdots,k,i}_{1,\cdots,k,j}$ denote the submatrix induced by rows 1 to k and i and columns 1 to k and j of A. Then,

$$a^k_{i,j} = \det\left((A)^{1,\cdots,k,i}_{1,\cdots,k,j}\right), k = \min(i,j) - 1. \tag{2}$$

As discussed in [23], the efficiency of IPGE can be improved by exploiting the zero pattern of sparse matrices. Specifically, if either $a^{k-1}_{k,j} = 0$ or $a^{k-1}_{i,k} = 0$ in the k-th iteration of IPGE, then (1) becomes $a^k_{i,j} = a^{k-1}_{i,j}\rho^k/\rho^{k-1}$.

Therefore, when a sequence (noted as $[\underline{k}..\overline{k}]$) of IPGE iterations exists such that for any $k \in [\underline{k}..\overline{k}]$, either $a^{k-1}_{k,j} = 0$ or $a^{k-1}_{i,k} = 0$, then

$$a^{\overline{k}}_{i,j} = \frac{\rho^{\overline{k}}}{\rho^{\overline{k}-1}}\frac{\rho^{\overline{k}-1}}{\rho^{\overline{k}-2}}\cdots\frac{\rho^{\underline{k}+1}}{\rho^{\underline{k}}}\frac{\rho^{\underline{k}}}{\rho^{\underline{k}-1}}a^{\underline{k}-1}_{i,j} = \frac{\rho^{\overline{k}}}{\rho^{\underline{k}-1}}a^{\underline{k}-1}_{i,j}. \tag{3}$$

Effectively, this allows to skip (combine) all the intermediate operations in a single operation (3), termed *History Update*.

B. Exact LU/Cholesky Factorization

Building on IPGE, [18] showed how to factorize a given (full rank) matrix as $A = LDU$, where $L = \{l_{i,j}\}$ is a lower-triangular matrix, $U = \{u_{i,j}\}$ is an upper-triangular matrix, and $D = \{d_{i,j}\}$ is a diagonal matrix. Specifically, each nonzero entry of L, U and D can be obtained as follows.

$$l_{i,j} = a^{j-1}_{i,j} \qquad \forall i \geq j, \tag{4}$$

$$d_{i,i} = 1/(\rho^{i-1}\rho^i) = 1/(a^{i-2}_{i-1,i-1}a^{i-1}_{i,i}), \tag{5}$$

$$u_{i,j} = a^{i-1}_{i,j} \qquad \forall i \leq j. \tag{6}$$

Moreover, when A is a non-singular symmetric positive-definite matrix, we have $U = L^T$, and thus the REF Cholesky factorization of A can be obtained as $A = LDL^T$ [18].

Let the frame matrix $F = \{f_{i,j}\}$ be defined as (7). Then, for the exact LU factorization $A = LDU$, L corresponds to the lower triangular section of F (denoted as $L = \text{tril}(F)$), while U is the upper triangular section of F (denoted as $U = \text{triu}(F)$), and the diagonals of F are the pivots ρ, from which we can compute D. As shown in (7), F is split into *frames* or groups of entries computed in the same IPGE iteration, which is noted by the superscript (denoted as κ). Note that $f_{i,j} = a^{\kappa}_{i,j}$ and that in iteration k only frames with index $\kappa > k$ need to be updated.

$$F := \begin{array}{|cccccc|c} a^0_{1,1} & a^0_{1,2} & \cdots & a^0_{1,k} & \cdots & a^0_{1,n} & \kappa = 0 \\ a^0_{2,1} & a^1_{2,2} & \cdots & a^1_{2,k} & \cdots & a^1_{2,n} & \kappa = 1 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ a^0_{k,1} & a^1_{k,2} & \cdots & a^{k-1}_{k,k} & \cdots & a^{k-1}_{k,n} & \kappa = k-1 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ a^0_{n,1} & a^1_{n,2} & \cdots & a^{k-1}_{n,k} & \cdots & a^{n-1}_{n,n} & \kappa = n-1 \end{array} \tag{7}$$

When A is factorized as $A = LDU$, solving $A\mathbf{x} = \mathbf{b}$ is done via specialized REF forward and backward substitution algorithms. Most importantly for this paper, the forward substitution process, $LD\mathbf{y} = \mathbf{b}$, can be performed without explicitly computing D and instead by performing n iterations of the following equation with $y^0_i = b_i$ [9], [11]:

$$y^k_i = \begin{cases} y^{k-1}_i & i \leq k, \\ \frac{y^{k-1}_i\rho^k - l_{i,k}y_k}{\rho^{k-1}} & i > k. \end{cases} \tag{8}$$

C. Backtracking

To update the exact factorization for column/row replacement, a new *push-and-swap* algorithm was proposed in [18] for dense matrices. The core of the push-and-swap approach is that the to-be-replaced column, k, is "pushed out" of the frame matrix by swapping it sequentially with the column to its right (initially, column $k+1$). A core operation of this algorithm is that the IPGE Update (1) for some entries must be "undone" via an operation referred to as *backtracking IPGE Update* (9).

Theorem 3 (Backtracking): For $k \in [1..\min(i,j)-1]$, given $a_{i,j}^{k}$, one can obtain without roundoff error the IPGE entry in the preceding iteration, $a_{i,j}^{k-1}$, as follows:

$$a_{i,j}^{k-1} = \frac{a_{i,j}^{k}\rho^{k-1} + a_{k,j}^{k-1}a_{i,k}^{k-1}}{\rho^{k}}. \tag{9}$$

The backtracking process is made more efficient for sparse matrices by combining it with the History Update (3). Specifically, consider the case when either $a_{k,j}^{k-1} = 0$ or $a_{i,k}^{k-1} = 0$, then the backtracking operation (9) becomes $a_{i,j}^{k-1} = a_{i,j}^{k}\rho^{k-1}/\rho^{k}$. Therefore, if there exists a sequence (noted as $[\underline{k}..\overline{k}]$) of backtracking such that for any $k \in [\underline{k}..\overline{k}]$, either $a_{k,j}^{k-1} = 0$ or $a_{i,k}^{k-1} = 0$, then

$$a_{i,j}^{\underline{k}-1} = \frac{\rho^{\underline{k}-1}}{\rho^{\underline{k}}}\frac{\rho^{\underline{k}}}{\rho^{\underline{k}+1}}\cdots\frac{\rho^{\overline{k}-2}}{\rho^{\overline{k}-1}}\frac{\rho^{\overline{k}-1}}{\rho^{\overline{k}}}a_{i,j}^{\overline{k}} = \frac{\rho^{\underline{k}-1}}{\rho^{\overline{k}}}a_{i,j}^{\overline{k}}. \tag{10}$$

Effectively, this allows to skip (combine) all the intermediate backtracking operations in a single operation (10), termed *backtracking History Update*.

D. Sparse Exact LU Update for Column Replacement

This section presents an algorithm to solve the following problem: Given a sparse matrix A, its (sparse) exact factorization $A = LDU$ (stored in frame matrix F), and a new column $\mathbf{v} \in \mathbb{Z}^n$ to replace the k-th column of A, obtain the exact factorization of the matrix obtained by replacing the k-th column of A with $\mathbf{v}$. For this purpose, we first give a high-level summary of the push-and-swap approach, which is the algorithm devised in [18] to solve the dense version of our problem.

Note that the to-be-replaced column k is unrelated to columns $1,..,k-1$ of F, is intimately related to column k of F, and an entry from it was used as a pivot to compute columns $k+1,...,n$ of F. The key idea of the (dense) push-and-swap approach is that column k is pushed out of the frame matrix by swapping it sequentially with the column to its right (initially, column $k+1$). Such adjacent column swaps are done in such a way that the frame matrix obtained is identical to the factorization that one would have obtained by factorizing the permuted matrix, but using some shortcuts expediting the computations: notably, frames $k+2,...,n$ only change sign when columns k and $k+1$ are swapped (a special case of Theorem 8). The core operation in the push-and-swap approach is the aforementioned backtracking operation. Finally, when the leaving column is in the last position, it can be swapped with the solution of $LD\mathbf{x} = \mathbf{v}$ (i.e., the incoming vector $\mathbf{v}$ after performing a REF forward substitution on it).

The key idea of our sparse exact push-and-swap approach is that, when the matrix A (and thus its factorization) is sparse, then pushing the to-be-replaced column k does not need to be done by swapping it sequentially with the next column to its right; instead column k can "leap" several columns simultaneously (as opposed to only one column) by exploiting the zero pattern. Depending on the zero pattern, the frame matrix resulting from the leap is computed with one of the algorithms presented next: the diagonal permutation pivot update (DPPU) and the column permutation pivot update (CPPU) algorithms.

The DPPU algorithm is used when the submatrix of F has the pattern shown in (11) or (12). The CPPU algorithm is applied when the submatrix of F has the pattern shown in (13), where $k' \in [k+1..n]$ is the column index of the next nonzero in the k-th row of F. For better illustration, the original and new k-th pivots in (11)-(13) are all enclosed in boxes.

$$(F)_{k,k+1,\cdots,k'}^{k,k+1,\cdots,n} = \begin{bmatrix} \boxed{a_{k,k}^{k-1}} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \boxed{a_{k',k'}^{k'-1}} \\ \vdots & \ddots & \vdots \end{bmatrix} \tag{11}$$

$$(F)_{k,k+1,\cdots,k'}^{k,k+1,\cdots,n} = \begin{bmatrix} \boxed{a_{k,k}^{k-1}} & \cdots & \cdot \\ 0 & \ddots & 0 \\ 0 & 0 & \boxed{a_{k',k'}^{k'-1}} \\ 0 & \ddots & \vdots \end{bmatrix} \tag{12}$$

$$(F)_{k,k+1,\cdots,k'}^{k,k+1,\cdots,n} = \begin{bmatrix} \boxed{a_{k,k}^{k-1}} & 0 & 0 & \boxed{a_{k,k'}^{k-1}} \\ \cdot & a_{k+1,k+1}^{k} & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix} \tag{13}$$

Next, we present the foundational theorems for the sparse exact LU factorization update algorithm for column replacement.

Theorem 4 (Backtracking in DPPU): Let A be a non-singular matrix such that the submatrix of its frame matrix F has the zero pattern in (11) or (12). Then, for $i,j \in [k+1..n]$,

$$a_{i,k'}^{k-1} = \frac{\rho^{k-1}}{\rho^{\kappa}} f_{i,k'}, \tag{14}$$

$$a_{k',j}^{k-1} = \frac{\rho^{k-1}}{\rho^{\kappa}} f_{k',j}. \tag{15}$$

Especially, for $i,j \in [k+1..k'-1]$,

$$a_{i,k'}^{k-1} = a_{k',j}^{k-1} = 0. \tag{16}$$

Theorem 5 (Diagonal-wise Switch of Originating Pivot (DwSOP)): Let F' be the updated frame matrix obtained by swapping columns and rows k and k' of F, whose submatrix follows the zero pattern in (11) or (12). Then we have $\rho^k := a_{k,k}^{k-1}$ and $\rho'^k := a_{k',k'}^{k-1}$. Moreover, for all entries in frames k to $k'-2$ but not the entries in the k'-th column or row, i.e., $\mathbb{S} := \{(i,j)|i,j \in [k+1..n]\backslash\{k'\}, \min(i,j) \leq k'-1\}$,

$$f'_{i,j} = \frac{\rho'^k}{\rho^k} f_{i,j}, (i,j) \in \mathbb{S}. \tag{17}$$

Theorem 6 (DPPU shortcuts): Let F' be the updated frame matrix obtained by swapping columns and rows k and k' of F, whose submatrix follows the zero pattern in (11) or (12). For $i,j \geq k'+1$ (i.e., frames k' to $n-1$),

$$f'_{i,j} = f_{i,j}. \tag{18}$$

Theorem 7 (Extended Row-wise Switch of Originating Pivot (xRwSOP)): Let F' be the updated frame matrix obtained by swapping columns k and k' of F, whose submatrix has the pattern shown in (13). Then, we have $\rho^k = f_{k,k}$ and $\rho'^k = f_{k,k'}$. In addition, for $i \in [k+1..n], j \in [k+1..k'-1]$, $f'_{i,j}$ can be equivalently obtained without roundoff error via

$$f'_{i,j} = \frac{\rho'^k}{\rho^k} f_{i,j} \tag{19}$$

Moreover, for $i \in [k+1..k'], j \in [k'+1..n]$, $f'_{i,j}$ can be equivalently obtained without roundoff error via the formula

$$f'_{i,j} = \frac{f_{i,j}\rho'^k - f_{i,k'} f_{k,j}}{\rho^k} \tag{20}$$

Theorem 8 (CPPU shortcuts): Let F' be the frame matrix obtained by swapping columns k and k' of F, whose submatrix has the pattern shown in (13). Then for $i,j \in [k'+1..n]$

$$f'_{i,j} = -f_{i,j}. \tag{21}$$

In addition, for the entries in the k'-th column of F' with row index $i \in [k+1..n]$,

$$f'_{i,k'} = -f_{i,k'}. \tag{22}$$

Fig. 1 illustrates the DPPU and CPPU algorithms, which are built on the above theorems. Due to space limitations and for simplicity, Fig. 1a only illustrates DPPU for updating F when its submatrix has the pattern (11); the process of updating F when its submatrix has the pattern (12) requires additional computations (specifically, to perform iterations of IPGE to k-th row of F to obtain row k' of F'). Fig. 1b illustrates CPPU for the case where only exactly one of $f_{k+1,k}$ and $f_{k,k+1}$ is zero (when both are zero, then the update can be done more efficiently with DPPU). The sparse exact LU update algorithm for column replacement, together with the details of the CPPU and DPPU algorithms, are given as follows, where $F = \{f_{i,j}\}$ is the frame matrix being updated, and the sparse vector $\mathbf{v} \in \mathbb{Z}^n$ is the entering vector to replace the k-th column of A.

Column addition: Solve $LD\mathbf{a}_{n+1} = \mathbf{v}$ with (8), and add $\mathbf{a}_{n+1}$ as the $(n+1)$-st column of F.

Column pushes: Iteratively push k-th column of F to position n by performing the DPPU algorithm if the submatrix of F has pattern as either (11) or (12), and the CPPU algorithm otherwise.

DPPU algorithm (Partially illustrated in Fig. 1a):

- backtrack frame $k'-1$ by simply multiplying all entries by $\alpha = f_{k-1,k-1}/f_{k'-1,k'-1}$ (Theorem 4);
- swap columns and rows of indices k and k';
- scale all entries between frames $k-1$ and $k'-1$ by $\lambda = f_{k,k}/f_{k',k'}$ (since entries are swapped) according to Theorem 7.
- perform IPGE for frame $k'-1$ (by multiplying all entries by $\lambda/\alpha = f_{k'-1,k'-1}/f_{k-1,k-1}$ if the submatrix of F follows pattern in (11)).

CPPU algorithm (Illustrated in Fig. 1b):

- Backtrack column k' and update the entries with row index $i \in [k+1..n]$ (in purple) in column k with the backtracking results;
- Swap column k and k' for entries with row index $i \in [1..k]$ (in yellow);
- Perform scaling for entries with column index $j \in [k+1..k'-1]$ and row index $i \in [k'+1..n]$ (in green) by $\lambda = f_{k,k}/f_{k,k'}$ (since entries are swapped) according to (19) in Theorem 7. Moreover, update entries in the red box according to (20);
- Flip sign for all entries in blue box according to CPPU shortcuts.

Column swap and deletion: Once the exiting column is pushed to the n-th position, swap it with the entering column (column $n+1$ of F) and remove it to obtain the final updated frame matrix.

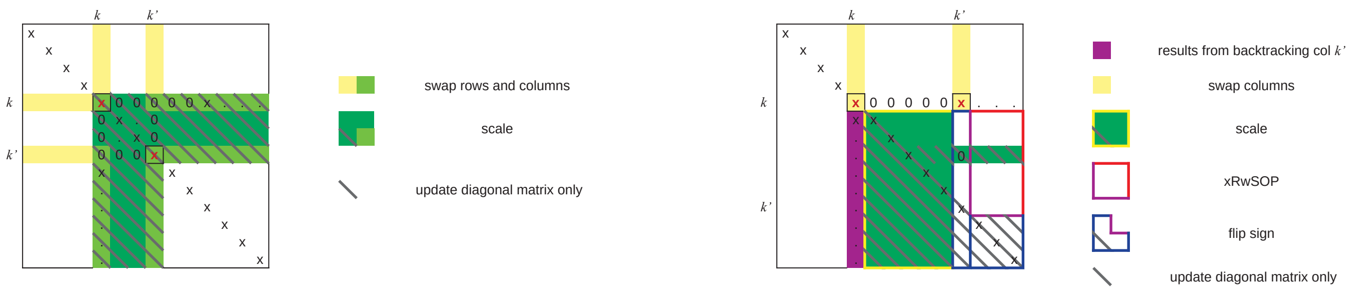
E. Sparse Exact Cholesky Update for Symmetric Column and Row Replacement

This algorithm updates L in the exact Cholesky factorization when the k-th column and row of A are symmetrically replaced (recall D can be obtained from L). This algorithm is similar to the one in Subsection II-D, but to maintain symmetry, only the DPPU algorithm is used. Specifically, the DPPU algorithm applies when the submatrix of F has pattern (11) or (23). Fig. 1a depicts how DPPU is used when F has pattern (11); when F has pattern (23), modifications are needed. Specifically, the last iteration of backtracking for column k' of L does not satisfy the requirement for (10), and thus the last iteration is done separately using (9), and the first iteration of IPGE for column k of L needs to be done separately with (1).

$$(F)^{k,k+1,\cdots,k'}_{k,k+1,\cdots,k'} = \begin{bmatrix} \boxed{a^{k-1}_{k,k}} & 0 & a^{k-1}_{k,k'} \\ 0 & \ddots & 0 \\ a^{k-1}_{k',k} & 0 & \boxed{a^{k'-1}_{k',k'}} \end{bmatrix} \tag{23}$$

F. Sparse Exact Cholesky Rank-1 Update/Downdate

The sparse exact Cholesky rank-1 update/downdate computes the exact Cholesky factorization of $\bar{A} = A + \sigma \mathbf{w}\mathbf{w}^T$ by modifying that of sparse matrix A, where $\mathbf{w} \in \mathbb{Z}^n$ is a sparse vector, and σ is either 1 (update) or -1 (downdate). The following theorem is the basis for this algorithm. Throughout


k
k'
swap rows and columns
scale
update diagonal matrix only
results from backtracking col k'
swap columns
scale
xRwSOP
flip sign
update diagonal matrix only

(a) DPPU algorithm if the submatrix of F has pattern (11) (b) CPPU algorithm

Fig. 1: The two sub-algorithms used in sparse exact LU update for column replacement. All entries marked as x must be nonzero. For all the shaded entries, leave $f_{i,j}$ untouched but update the j-th diagonal of D_L or i-th diagonal of D_U correspondingly.

this subsection, denote the exact Cholesky factorization of A given as LDL^T, and that of $\bar{A} = A + \sigma \mathbf{w}\mathbf{w}^T$ as $\bar{L}\bar{D}\bar{L}^T$.

Theorem 9 (Exact Cholesky rank-1 update/downdate): When using (8) to solve $LD\mathbf{x} = \mathbf{w}$ or $\bar{L}\bar{D}\bar{\mathbf{x}} = \mathbf{w}$, the following equality holds for $k \in [0..n]$,

$$x_i^k = \bar{x}_i^k. \tag{24}$$

In addition, let $\bar{\rho}^i = \bar{l}_{i,i}$, then

$$\bar{l}_{i,j} = l_{i,j}\frac{\bar{\rho}^{j-1}}{\rho^{j-1}} + \sigma x_j^{j-1}\frac{x_i^{j-1}}{\rho^{j-1}}. \tag{25}$$

G. Sparse Exact LU Rank-1 Update/Downdate

The sparse exact LU rank-1 update/downdate is an extension of the sparse exact Cholesky rank-1 update/downdate. It is used when the change to A is not symmetric; i.e., when $\bar{A} = A + \sigma \mathbf{u}\mathbf{v}^T$. The following theorem shows how to obtain the exact factorization of $\bar{A}$ by updating the factorization of A. Throughout this subsection denote the exact LU factorization of A as LDU, and that of $\bar{A} = A + \sigma \mathbf{u}\mathbf{v}^T$ as $\bar{L}\bar{D}\bar{U}$.

Theorem 10 (Exact LU rank-1 update/downdate): When using (8) to solve $LD\mathbf{x} = \mathbf{u}$ or $\bar{L}\bar{D}\bar{\mathbf{x}} = \mathbf{u}$, and $U^TD\mathbf{y} = \mathbf{v}$ or $\bar{U}^T\bar{D}\bar{\mathbf{y}} = \mathbf{v}$, the following equality holds for $k \in [0..n]$,

$$x_i^k = \bar{x}_i^k, y_i^k = \bar{y}_i^k. \tag{26}$$

In addition, let $\bar{\rho}^i = \bar{l}_{i,i} = \bar{u}_{i,i}$, then

$$\bar{l}_{i,j} = l_{i,j}\frac{\bar{\rho}^{j-1}}{\rho^{j-1}} + \sigma x_j^{j-1}\frac{x_i^{j-1}}{\rho^{j-1}}, \tag{27}$$

$$\bar{u}_{j,i} = u_{j,i}\frac{\bar{\rho}^{j-1}}{\rho^{j-1}} + \sigma y_j^{j-1}\frac{y_i^{j-1}}{\rho^{j-1}}. \tag{28}$$

III. Data Structure and Implementation

This section discusses the data structures and efficient implementations of the update algorithms.

A. Data Structures

For the sparse matrices in all four update algorithms, we store A, L, and U^T as an array of sparse column vectors, where each column vector is independently allocated, thereby allowing its size to grow or shrink without affecting other columns. This approach has two advantages: (i) since all algorithms update the frame matrix F frame-wise (i.e., update one or multiple frames per iteration), we can update column(s) of L and U^T in the same frame(s) as needed and (ii) when the number of entries in one column of L or U^T changes, there is no need to update other columns. In each column vector, the first entry is the diagonal entry; however the indices of the remaining entries are unsorted. Thus, the addition of fill-in to a column or the deletion of entry from a column can be done very efficiently. All the sparse exact factorization update algorithms are implemented using the GNU Multiple Precision Arithmetic (GMP) Library [24]. Specifically, all entries are stored as full-precision integers (i.e., `mpz_t`).

B. Implementation

There are three types of operations involved in all sparse exact update algorithms. The first type of operation updates an entry by multiplying with a rational scalar (i.e., $f'_{i,j} = f_{i,j}t/s$), such as equations in Theorems 4 and 8, and (19). The second type performs an IPGE-like update in form of $f'_{i,j} = (f_{i,j}t - uv)/s$, which includes the IPGE update (1) and (20) in Theorem 7. The third type updates an entry with a backtracking-like equation in form of $f'_{i,j} = (f_{i,j}t + uv)/s$, which includes the backtracking operation (9) and the rank-1 update (i.e., (25), (27) and (28)). In fact, the second and third types of updates can become pure scaling under certain condition(s). For example, when $f_{i,k'} = 0$ or $f_{k,j} = 0$, (20) becomes exactly the same as (19), or when $x_j^{j-1} = 0$, (25) becomes $\bar{l}_{i,j} = l_{i,j}\bar{\rho}^{j-1}/\rho^{j-1}$. Therefore, when thinking of the algorithms as updating vectors (as opposed to single entries), most involve multiplying all entries in a frame or a column/row of a frame by a common rational scalar. To further improve the efficiency of the update algorithm by postponing these scalar-vector multiplications, we introduce two additional diagonal rational matrices, D_L and D_U, such that $L = \text{tril}(F) * D_L$ and $U = D_U * \text{triu}(F)$ (initially, $D_L = D_U = I$). Then, in the case when all entries in the j-th column (i-th row) of a frame need to by multiplied by a common scaling factor, we only update the j-th (i-th) diagonal of D_L (D_U).

Given D_L and D_U, all shaded entries in Fig. 1 can be updated without explicitly performing the multiplication. Instead, only D_L and/or D_U need to be updated. For Fig. 1b, all the entries in the row of the $(i_0 - 1)$-st frame can be updated via

(19) and thus the above technique can also be applied, where $i_0 \in [k+1..k']$ is the row index such that $f_{i_0,k'} = 0$.

The implicit update for the j-th column of L by updating the diagonals of D_L can remain implicit if the next update for the j-th column of L still requires only pure multiplication. The same will hold for the i-th row of U. Moreover, the backward substitution phase of solving $A\mathbf{x} = \mathbf{b}$ (i.e., solving $U\mathbf{x} = \mathbf{y}$) does not require any entries of U to be explicitly updated. Instead, x'_i can be solved as

$$x'_i = \frac{y_i \rho^n - D_U(i,i) \sum_{j=i+1}^{n} u_{i,j} x'_j}{\rho^i}$$

where $\mathbf{x}' = \mathbf{x}\rho^n$ [9], $D_U(i,i)$ is the i-th diagonal of D_U and the pending scaling factor for i-th row of U.

There are cases when the entries need to be explicitly updated (and the corresponding diagonal of D_L and/or D_U is then reset to 1): (i) when the explicit factorization $A = LDU$ is desired/required; (ii) when the j-th column of L is needed to perform the j-th iteration of IPGE update for given vector or the forward substitution when solving $LD\mathbf{y} = \mathbf{b}$, in which all entries in that column of L need to be multiplied by $D_L(j,j)$; and (iii) when the entries in a row/column of a frame needs to be partially/entirely updated with the second or third aforementioned update (i.e., $f'_{i,j} = (f_{i,j}t \pm uv)/s$). In these cases, these updates can also be efficiently done because both the operand and the result of such updates are guaranteed to be in integer domain.

The explicit update for case (ii) is needed due to the following two reasons. The first reason is the use of (3) for successive IPGE updates in the sparse algorithm. If only one iteration of IPGE update is needed, the implicit update can theoretically remain implicit. Consider the j-th column of L is used to perform the j-th IPGE update for $\mathbf{x}$, and $D_L(j,j) \neq 1$, then $\rho^j = l_{j,j} D_L(j,j)$ and the update for $x_i, i > j$ should be

$$x_i = \frac{x_i \rho^j - x_j l_{i,j} D_L(j,j)}{\rho^{j-1}} = \frac{x_i l_{j,j} - x_j l_{i,j}}{\rho^{j-1}} D_L(j,j).$$

According to this equation, x_i can be updated without applying or changing $D_L(j,j)$. However, when a sequence of IPGE updates is performed and (3) is used, the entries will not be correctly updated. Furthermore, updating x_i without applying any pending scaling factor $D_L(j,j) \neq 1$ could result in a non-integer value of x_i, which breaks the exact factorization framework. Thus, the major reason for explicit update for case (ii) is to keep the resulting entry in the integer domain.

Consider the case where an entry $f_{i,j}$ needs to be explicitly updated by multiplying with a rational number t_1/t_2, $t_1, t_2 \in \mathbb{Z}, \gcd(t_1,t_2) = 1$. For example, in Fig. 1b, all the green entries in the row of the $(i-1)$-th frame must be explicitly updated with $\lambda = f_{k,k}/f_{k,k'}$, where $i \in [k+1..k']$ is the row index such that $f_{i,k'} \neq 0$. In this case, since λ remains the same for all green entries in the same row, it is computed and stored as a full-precision rational number (as `mpq_t`). For such case, instead of performing $f_{i,j} = f_{i,j} * t_1$ and $f_{i,j} = f_{i,j}/t_2$ sequentially, the computation can be done in two steps as $f_{i,j} = f_{i,j}/t_2$ followed by $f_{i,j} = f_{i,j} * t_1$. Both computation sequence can guarantee the final result, as well as the intermediate result, to be in integer domain (either division therein is guaranteed to have zero remainder). However, the latter computation sequence will be relatively faster since the first method increases the size of $f_{i,j}$ before the second step.

Consider a different case in which $f_{i,j}$ needs to be updated but it was implicitly updated with the scaling factor γ stored as either $D_L(j,j)$ or $D_U(i,i)$. Depending on the update to be performed on $f_{i,j}$, we have three scenarios:

- If $f_{i,j}$ needs to be updated explicitly by multiplying with a rational scalar t_1/t_2, we first compute $\gamma t_1/t_2$ by removing any common factor between denominator and numerator. Then follow the procedure discussed above.
- If $f_{i,j}$ needs to be updated with an IPGE-like formula (i.e., $f'_{i,j} = (f_{i,j}t - uv)/s$), we compute $f'_{i,j} = \gamma' f_{i,j} - uv/s$ with $\gamma' = \gamma t/s$ instead of directly solving $f'_{i,j} = (\gamma f_{i,j} t - uv)/s$. Since the value of t/s is the same for all entries in the same column/row of a frame, $\gamma' = \gamma t/s$ can be always computed by removing any common factor between denominator and numerator in advance. Then $f'_{i,j} = \gamma' f_{i,j} - uv/s$ involves 2 multiplications, 2 divisions and 1 subtraction, while $f'_{i,j} = (\gamma f_{i,j} t - uv)/s$ requires 3 multiplication, 2 divisions and 1 subtraction. Therefore, computing γ' in advance accelerates this update. Since the update only guarantees $f'_{i,j}$, but not $\gamma' f_{i,j}$ or uv/s, to be an integer, we can instead compute $f'_{i,j}$ as `floor`$(\gamma' f_{i,j})$ - `floor`(uv/s).
- If $f_{i,j}$ needs to be updated with a backtracking-like formula (i.e., $f'_{i,j} = (f_{i,j}t + uv)/s$), we compute $f'_{i,j} = \gamma' f_{i,j} + uv/s$ with $\gamma' = \gamma t/s$ instead of directly solving $f'_{i,j} = (\gamma f_{i,j} t + uv)/s$. Again, $\gamma' = \gamma t/s$ can be always computed in advance for a whole row/column of a frame, and $f'_{i,j} = \gamma' f_{i,j} + uv/s$ involves one less multiplication. Therefore, knowing that the result must be an integer, it is more efficient to compute the update as $f'_{i,j} = \gamma' f_{i,j} + uv/s =$ `floor`$(\gamma' f_{i,j})$ + `ceil`(uv/s).

where the divisions of `floor` and `ceil` are done in exact arithmetic using `mpz_fdiv_q` and `mpz_cdiv_q`.

IV. Computational Tests and Analysis

This section analyzes the performance of the algorithms for the sparse exact LU update for column replacement and the sparse exact Cholesky rank-1 update/downdate.

A. Computing Environment and Test Instances

All of the tests were run on an IBM Power8 with 1TB of RAM with 20 hardware cores (4GHz, dual socket). Only a single core was used for these experiments. To illustrate the performance in real-world applications, the sparse LU column replacement update and the sparse Cholesky rank-1 update/downdate were used in the linear programming simplex and dual active set algorithms, respectively. We tested on matrices from the *LPnetlib* group within the SuiteSparse Matrix Collection [19]. Of the 138 problems from the *LPnetlib* group, 29 were infeasible and 31 were optimal from the initial basis or too simple (recorded running time is 0 second) or too large to solve. Thus we restrict our comparison to the remaining 78 instances. Note that all input matrices are processed such that all entries are integer, all fixed variables (with same upper and lower bounds) are removed, and all lower bounds are zero.

B. Results for Sparse Exact LU Column-Replace Update

While the simplex algorithm allows the existence of nonzero values for the non-basic variables, our simple implementation of simplex algorithm assumes that all non-basic variables are zero. Therefore, we use the GLPK library [25] to search for an initial basis set that meets this assumption. During each iteration of our implementation of the simplex algorithm, we solve sparse linear equations exactly to find the entering and exi iting columns for the basis matrix and obtain the exact factorization for the new basis matrix from both direct LU factorization (DLU) and iteratively LU update (LUU).

For each of the 78 cases, we ran simplex for up to 100 iterations to find an optimal basis. Fig. 2 shows the total time of DLU and LUU for at most 100 iterations. The time ratio of DLU over LUU for each case ranges from 1.066 (`lp_osa_07`) to 228.5 (`lp_ship12l`). Fig. 2 shows that LUU outperforms DLU in all cases and the improvement is more significant for larger problems.

DLU always finds a permutation which reduces fill-in and bit-length of entries, however, the LUU modification may lead to an undesirable permutation. Therefore, iteratively using LUU may lead to excessive fill-in or entries of large bit-length. Thus, we next show the effect of permutation on the performance of the updates by performing the update based on the direct factorization of the previous basis matrix, which can be considered as a rough estimate of the lower-bound of the update algorithm and thus denoted as lb.

Fig. 3 shows the results of DLU, LUU and the so-called lb for case `lp_osa_07`, which is the case that the average time for DLU is closest to that of LUU. Specifically, Fig. 3c shows the time for searching the entering and existing columns by exactly solving sparse linear equations based on the factorization from LUU, from which one can see that several significant jumps occur in the 48th, 54th and 76th iterations. Moreover, similar jumps happen in the results of LUU in Figs. 3b, 3e and 3f. However, the results of lb do not show this kind of pattern, which indicates that the quality of the permutation after LUU has dropped significantly during these iterations and needs to be reset with the help of DLU to have better performance. This could be the next direction of improvement for future work on these algorithms.

C. Results for Sparse Exact Cholesky Rank-1 Up/Down-date

This subsection presents the performance of the sparse exact Cholesky rank-1 update/downdate algorithm. Let B denote the basis matrix obtained via the GLPK library. Then, from the remaining columns associated with the non-basic variables, we select one of the sparsest vectors $\mathbf{u}$ and one of the densest vectors $\mathbf{v}$. Then we compute $A_0 = BB^T$ and its exact Cholesky factorization. Next, we find the exact Cholesky factorizations for $A_1 = A_0 + \mathbf{u}\mathbf{u}^T$ and $A_2 = A_1 + \mathbf{v}\mathbf{v}^T$

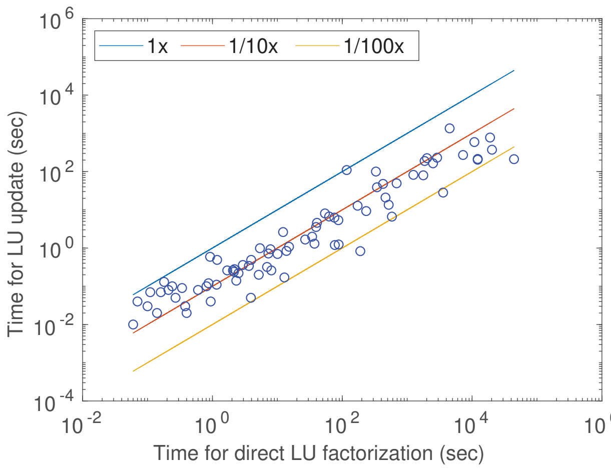


Fig. 2: Performance of sparse exact LU factorization update for column replacement, compared with direct LU factorization.

via direct factorization and rank-1 update. Similarly, we find the factorizations for $A_3 = A_2 - \mathbf{v}\mathbf{v}^T = A_1$ and $A_4 = A_3 - \mathbf{u}\mathbf{u}^T = A_0$ via direct factorization and rank-1 downdate. For each matrix, Fig. 4 compares the total time of all the direct factorization to the total time of all rank-1 update/downdate. Fig. 4 does not include 11 out of the 78 cases because the recorded total time for either the direct factorization or the rank-1 update/downdate was 0 second and one case is too large to solve (`lp_fit2p`). The time ratio (direct factorization over rank-1 update/downdate) for each matrix ranges from 1.333 (`lp_agg2`) to 1698 (`lp_perold`). Furthermore, the speedups obtained by our update/downdate algorithm tend to be higher for larger problems.

V. Conclusion

This paper reviews the theoretical foundation and proposed algorithms for two kinds of updates for sparse exact Cholesky and LU factorization, namely the update for column and/or row replacement and rank-1 update/downdate. The paper summarizes the common operations used by these four update algorithms, and proposes a data structure to efficiently implement these algorithms. The computational results show that the update algorithms provide substantial performance gains over direct factorization. In the future, we will try to further improve the performance of the sparse exact LU column-replacement update by checking the quality of the permutation as updates are made and selectively re-factorizing as needed, and our finalized open-source code will be made publicly available.

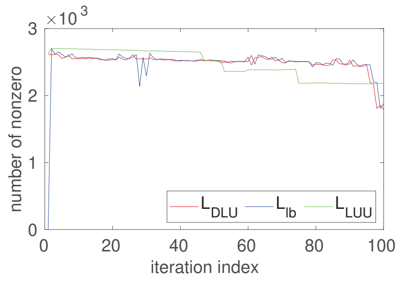
References

[1] R. H. Bartels and G. H. Golub, “The simplex method of linear programming using LU decomposition,” *Commun. ACM*, vol. 12, no. 5, p. 266–268, May 1969.

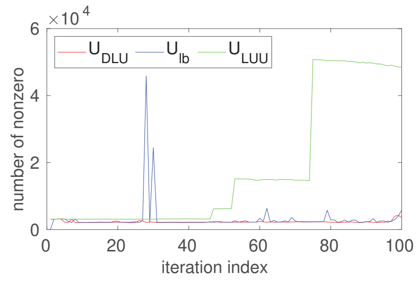
[2] D. Ma and M. A. Saunders, “Solving multiscale linear programs using the simplex method in quadruple precision,” in *Numerical analysis and optimization*. Springer, 2015, pp. 223–235.

[3] D. H. Bailey and J. M. Borwein, “High-precision arithmetic in mathematical physics,” *Mathematics*, vol. 3, no. 2, pp. 337–367, 2015.

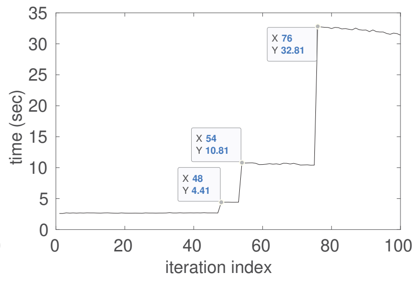
[4] S. A. Sarra, “Radial basis function approximation methods with extended precision floating point arithmetic,” *Engineering Analysis with Boundary Elements*, vol. 35, no. 1, pp. 68–76, 2011.



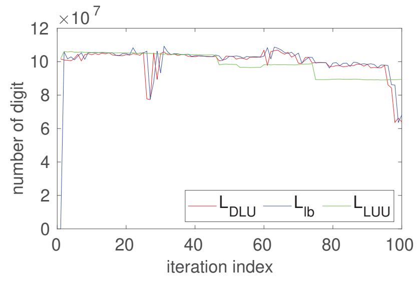
(a) number of nonzeros in L



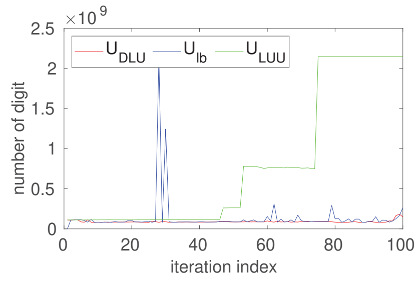
(b) number of nonzeros in U



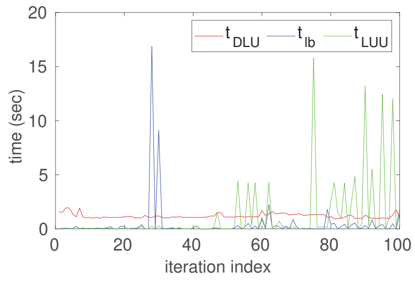
(c) time to search the entering and existing columns



(d) number of bits of all entries in L



(e) number of bits of all entries in U



(f) time for factorization or update

Fig. 3: The detailed performance results for `lp_osa_07`.

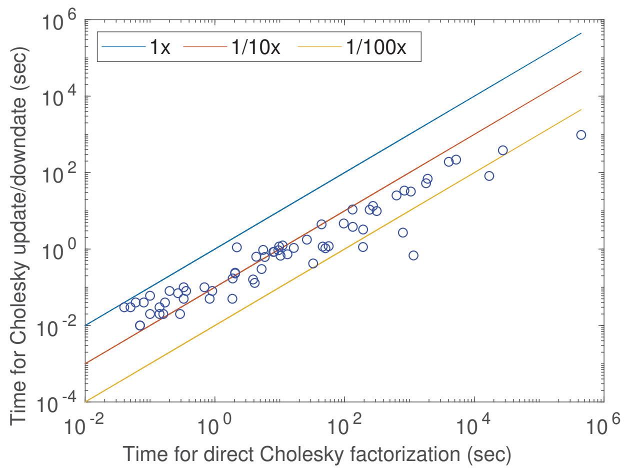


Fig. 4: Performance of sparse exact Cholesky rank-1 update/downdate, compared with direct Cholesky factorization.

[5] D. L. Applegate, W. Cook, S. Dash, and D. G. Espinoza, "Exact solutions to linear programming problems," *Operations Research Letters*, vol. 35, no. 6, pp. 693–699, 2007.

[6] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse *et al.*, "The SCIP optimization suite 7.0," ZIB, Takustr. 7, 14195 Berlin, Tech. Rep. 20-10, 2020.

[7] A. Gleixner, D. Steffy, and K. Wolter, "Iterative refinement for linear programming," ZIB, Takustr. 7, 14195 Berlin, Tech. Rep. 15-15, 2015.

[8] A. M. Gleixner, "Exact and fast algorithms for mixed-integer nonlinear programming," 2015.

[9] A. R. Escobedo and E. Moreno-Centeno, "Roundoff-error-free algorithms for solving linear systems via Cholesky and LU factorizations," *INFORMS Journal on Computing*, vol. 27, no. 4, pp. 677–689, 2015.

[10] A. R. Escobedo, E. Moreno-Centeno, and C. Lourenco, "Solution of dense linear systems via roundoff-error-free factorization algorithms: Theoretical connections and computational comparisons," *ACM Transactions on Mathematical Software (TOMS)*, vol. 44, pp. 1–24, 2018.

[11] C. Lourenco, A. R. Escobedo, E. Moreno-Centeno, and T. A. Davis, "Exact solution of sparse linear systems via left-looking roundoff-error-free LU factorization in time proportional to arithmetic work," *SIAM J. Matrix Analysis and Applic.*, vol. 40, pp. 609–638, 2019.

[12] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, "Methods for modifying matrix factorizations," *Mathematics of computation*, vol. 28, no. 126, pp. 505–535, 1974.

[13] J. A. Tomlin, "Modifying triangular factors of the basis in the simplex method," in *Sparse matrices and their applications*. Springer, 1972, pp. 77–85.

[14] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, "Maintaining LU factors of a general sparse matrix," *Linear Algebra and its Applications*, vol. 88, pp. 239–270, 1987.

[15] T. A. Davis and W. W. Hager, "Row modifications of a sparse Cholesky factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 3, pp. 621–639, 2005.

[16] L. M. Suhl and U. H. Suhl, "A fast LU update for linear programming," *Annals of Operations Research*, vol. 43, no. 1, pp. 33–47, 1993.

[17] T. A. Davis and W. W. Hager, "A sparse proximal implementation of the LP dual active set algorithm," *Mathematical programming*, vol. 112, no. 2, pp. 275–301, 2008.

[18] A. R. Escobedo and E. Moreno-Centeno, "Roundoff-error-free basis updates of LU factorizations for the efficient validation of optimality certificates," *SIAM Journal on Matrix Analysis and Applications*, vol. 38, no. 3, pp. 829–853, 2017.

[19] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.

[20] J. Edmonds, "Systems of distinct representatives and linear algebra," *J. Res. Nat. Bur. Standards Sect. B*, vol. 71, no. 4, pp. 241–245, 1967.

[21] E. H. Bareiss, "Sylvester's identity and multistep integer-preserving gaussian elimination," *Mathematics of computation*, vol. 22, no. 103, pp. 565–578, 1968.

[22] R. M. Montante-Pardo and M. A. Méndez-Cavazos, "Un método número para cálculo matricial," *Revista Técnico-Cientıfica de DFIMEUEANL, Nueov León*, vol. 2, pp. 1–24, 1977.

[23] H. R. Lee and B. D. Saunders, "Fraction free gaussian elimination for sparse matrices," *Journal of symbolic computation*, vol. 19, no. 5, pp. 393–402, 1995.

[24] T. Granlund, "GNU multiple precision arithmetic library," *http://gmplib.org/*, 2010.

[25] A. Makhorin, "GLPK (GNU linear programming kit)," *http://www. gnu. org/s/glpk/glpk. html*, 2008.