Solution of Dense Linear Systems via Roundoff-Error-Free Factorization Algorithms: Theoretical Connections and Computational Comparisons

ADOLFO R. ESCOBEDO, Arizona State University

ERICK MORENO-CENTENO and CHRISTOPHER LOURENCO, Texas A&M University

Exact solving of systems of linear equations (SLEs) is a fundamental subroutine within number theory, formal verification of mathematical proofs, and exact-precision mathematical programming. Moreover, efficient exact SLE solution methods could be valuable for a growing body of science and engineering applications where current fixed-precision standards have been deemed inadequate. This article contains key derivations relating, and computational tests comparing, two exact direct solution frameworks: roundoff-error-free (REF) LU factorization and rational arithmetic LU factorization. Specifically, both approaches solve the linear system $Ax = b$ by factoring the matrix A into the product of a lower triangular (L) and upper triangular (U) matrix, $A = LU$. Most significantly, the featured findings reveal that the integer-preserving REF factorization framework solves dense SLEs one order of magnitude faster than the exact rational arithmetic approach while requiring half the memory. Since rational LU is utilized for basic solution validation in exact linear and mixed-integer programming, these results offer preliminary evidence of the potential of the REF factorization framework to be utilized within this specific context. Additionally, this article develops and analyzes an efficient streamlined version of Edmonds's Q -matrix approach that can be implemented as another basic solution validation approach. Further experiments demonstrate that the REF factorization framework also outperforms this alternative integer-preserving approach in terms of memory requirements and computational effort. General purpose codes to solve dense SLEs exactly via any of the aforementioned methods have been made available to the research and academic communities.

CCS Concepts: • **Mathematics of computing** → **Solvers**; **Computations on matrices**; • **Computing methodologies** → **Exact arithmetic algorithms**; **Linear algebra algorithms**; **Factorization methods**;

Additional Key Words and Phrases: Exact linear programming, roundoff errors, dense linear systems

ACM Reference format:

Adolfo R. Escobedo, Erick Moreno-Centeno, and Christopher Lourenco. 2018. Solution of Dense Linear Systems via Roundoff-Error-Free Factorization Algorithms: Theoretical Connections and Computational Comparisons. *ACM Trans. Math. Softw.* 44, 4, Article 40 (June 2018), 24 pages.

<https://doi.org/10.1145/3199571>

1 INTRODUCTION

Exact solving of systems of linear equations (SLEs) is of fundamental importance for number theory (Beliakov and Matiyasevich 2016; Dixon 1982), formal verification of mathematical proofs

This article was supported by the National Science Foundation under Grant No. CMMI-1252456.

Authors' addresses: A. R. Escobedo, Arizona State University, P.O. Box 878809, Tempe, AZ 85287; email: adres@asu.edu; E. Moreno-Centeno, Texas A&M University, 3131 TAMU, College Station, TX 77843; email: emc@tamu.edu; C. Lourenco, Texas A&M University, 3131 TAMU, College Station, TX 77843; email: clouren@tamu.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 ACM 0098-3500/2018/06-ART40 \$15.00

<https://doi.org/10.1145/3199571>

(Solovyev and Hales 2011; Wiedemann 1986), and exact-precision mathematical programming (Cook et al. 2011; Dhiflaoui et al. 2003; Gleixner 2015; Steffy 2011a). Moreover, efficient exact SLE solution methods could be valuable for a growing body of applications where current fixed-precision standards have been deemed inadequate. These include the computation of radial basis functions for scattered data interpolation (Sarra 2011), studies of anharmonic oscillations in semiconductor engineering and mathematical physics (Bailey and Borwein 2015), and flux balance analysis for the study of biochemical reaction networks (Ma and Saunders 2015). For these and other modern contexts, extended precision is needed to accurately express very small yet meaningful solution values (e.g., of order 10^{-10}) and/or to mitigate the impact of inherently severe numerical issues. Indeed, the compounding of roundoff errors encountered while solving SLEs in fixed precision may lead mathematical software to produce erroneous solutions—particularly if an SLE is numerically ill-conditioned or large (requiring many operations to solve) (Klotz 2014). Therefore, it is imperative to devise efficient algorithms and robust software to account for, and ultimately eliminate, roundoff errors accrued while solving SLEs.

One essential need for the elimination of roundoff errors stems from the dramatic impact that they can have on the behavior of mathematical programming software. The accumulation of roundoff errors in LU factorization¹ subroutines may cause popular linear programming (LP) and mixed-integer programming (MIP) solvers to report suboptimal solutions as optimal (Dhiflaoui et al. 2003). For example, the distinct “optimal” objective values obtained when solving the real-world LP instance *sgpf5y6* (Mittelmann 2006) may deviate by as much as 3% relative to the true optimal solution, depending on the specific solver and solver-version utilized (Steffy 2011b); the deviation from this problem’s true optimal value actually worsened in a newer version of the same commercial solver (Escobedo 2016). The implications of unknowingly making suboptimal decisions that are based on invalid solutions could be severe and costly for many industries (Steffy 2011b). All the more concerning is the fact that roundoff errors can also cause LP and MIP solvers to misclassify feasible problems as infeasible and vice versa (Koch 2004; Neumaier and Shcherbina 2004). Other puzzling outcomes can unexpectedly occur and remain largely unreported based on their less clear classification; for instance, from the authors’ experiences solving large-scale power systems problems in Dehghanian et al. (2015) and Escobedo et al. (2014), roundoff errors could lead to a paradoxical outcome in which the working basis is eventually flagged as being singular. Though infrequent and unpredictable, the incidence of these incongruous outcomes brings the reliability of mathematical programming software into question and could expose users to serious risks. This is especially disquieting since the prescribed advanced modeling techniques and ad hoc solver settings for mitigating roundoff errors may still be insufficient for preventing the occurrence of such outcomes (Klotz 2014). In light of these observations, it is imperative to develop viable tools that can validate the output of LP and MIP solvers and that can be implemented to combat the deleterious effects of roundoff errors efficiently.

To help address these issues, the roundoff-error-free (REF) factorization framework, introduced in Escobedo and Moreno-Centeno (2015), solves rational SLEs efficiently without accruing roundoff errors. This framework consists of the REF LU and REF Cholesky factorizations and the REF forward and backward substitution algorithms, all of which rely solely on integer-preserving arithmetic. Recently, in Escobedo and Moreno-Centeno (2017) said framework was expanded to perform integer-preserving basis updates for the REF factorizations. Altogether, these tools can be used to exactly solve a variety of dense SLEs. In particular, they aim to advance operations research practice for two interrelated reasons: (1) solutions yielded by fixed-precision LP and MIP solvers come with certificates (i.e., primal and dual feasibility) that can be validated only approximately within

¹LU factorization is used to solve a SLE, $Ax = b$ by factoring the matrix A into the product $A = LU$ where L and U are lower and upper triangular matrices, respectively.

numerical tolerances and (2) state-of-the-art computationally exact versions of these solvers implement algorithms that rely on exact rational arithmetic, which involves performing recurring greatest common divisor (GCD) calculations and other expensive operations. Ultimately, the goal of the REF factorization framework is to provide fail-safe yet efficient validation subroutines for guaranteeing the dependability of mathematical programming software.

This article makes the following contributions. First, it establishes theoretical connections between the REF LU factorization and the Doolittle and Crout LU factorizations. Second, it provides cogent evidence that the REF factorization framework is computationally superior to the exact rational arithmetic LU factorization approach when the underlying SLEs are dense. Third, it derives an efficient adaptation of Edmonds's Q-matrices (Edmonds and Maurras 1997) for use as a basic solution validation subroutine. Fourth, it demonstrates that the REF factorization framework outperforms the adapted Edmonds's Q-matrix approach in this context in terms of required memory and computational effort for dense SLEs.

The article is organized as follows. Section 2 provides necessary background material for contextualizing this work, Section 2.1 outlines the two leading mixed-precision approaches utilized by exact LP and MIP solvers to obtain valid solutions, Section 2.2 summarizes previous computational tests of techniques for obtaining exact solutions to SLEs within the context of LP, and Section 2.3 reviews relevant components of the REF factorization framework. Section 3 outlines specifications for the computational tests on dense SLEs featured in this work. Section 4 presents analytical and computational comparisons between the REF LU factorization framework and the exact rational arithmetic LU factorization approach. Section 5 derives an efficient adaptation of Edmonds's Q-matrices for basic solution validation, and then it provides corresponding comparisons with the REF factorization framework. Section 6 concludes the article.

2 BACKGROUND

2.1 Mixed-Precision Approaches for Solving LPs Exactly

Implementing unlimited-precision LP and MIP solvers is computationally expensive and impractical (Applegate et al. 2007a; Cook et al. 2011; Gärtner 1999). Consequently, mixed-precision approaches combining floating-point operations and exact arithmetic have been developed for solving LPs exactly—which, due to the role of LP bounds, also implicates the valid solution of MIPs. Two early implementations with limited applicability were devised in Dhiflaoui et al. (2003) and Gärtner (1999). More recently, the QSopt_ex Rational LP Solver (Applegate et al. 2007b) adopted a precision-boosting approach and SoPlex (Gleixner et al. 2015; Wunderling 1996)—the standard LP solver in the SCIP Optimization Suite (Gamrath et al. 2016)—the LP iterative refinement approach; each is depicted in Figure 1 and outlined in the ensuing paragraphs.

Applegate et al. (2007a) developed the precision-boosting approach implemented within QSopt_ex and illustrated in Figure 1(a). In this approach, the floating-point simplex algorithm is first performed in native precision α_0 , after which an exact rational arithmetic LU factorization of the final basis is constructed to validate the solver's output certificate—e.g., optimality, infeasibility. When the latter subroutine invalidates the output, the working floating-point precision α is increased by $\Delta > 0$ and the LP solver resumes from the last known basis; the resulting higher-precision solution is in turn put through LU basis validation. The process repeats until the solver output certificate is validated.

Gleixner et al. (2012, 2016) extended the technique of iterative refinement for solving SLEs (see Wilkinson (1963)) to LP. Their approach calculates solutions of arbitrary precision by solving a sequence of closely related LPs in native precision, with each successive one computing a correction of the previous problem's approximate solution, until the error falls below a specified tolerance. Since it relies on fixed-precision simplex runs, however, LP iterative refinement is susceptible

Fig. 1. Leading mixed-precision approaches for solving LPs exactly.

to nontrivial roundoff errors and may require a series of different parameter adjustments to run successfully on numerically difficult instances (Gleixner et al. 2016). To guarantee a valid/exact LP solution, its implementation in SoPlex can be paired with LU basis validation, which is triggered when the basis does not change for a preset number of refinement iterations $\tilde{K}$. This mixed-precision approach is depicted in Figure 1(b); therein, the LU basis validation subroutine mirrors that of Figure 1(a), but its contents are shown in collapsed form for succinctness.

As Figure 1 illustrates, both mixed-precision approaches compartmentalize exact computation from the floating-point simplex runs. That is to say, if the LU basis validation subroutines were to be replaced by more efficient algorithms for obtaining exact primal and dual solutions, the inputs and outputs of the respective LP solves would be unaffected. Hence, making major improvements to LU basis validation, which is precisely what the REF factorization framework seeks to accomplish, would represent significant advances to exact LP. This would also impact exact MIP since valid LP bounds depend on valid LP output; in fact, the exact MIP solver devised in Cook et al. (2011, 2013) employs QSOpt_ex for this purpose. Improving the performance of both state-of-the-art implementations featured in Figure 1 is worthwhile because each offers distinctive computational advantages, as attested by the study by Gleixner (2015) involving 1,202 primal and dual feasible LPs drawn from various standard repositories. Therein, SoPlex paired with LU basis validation was 1.85 to 3 times faster than the precision-boosting QSOpt_ex implementation; however, QSOpt_ex was better suited to handle numerically difficult problems, as evinced by its successful solution of six of seven such problems that could not be solved via the SoPlex implementation. It is important to acknowledge that, since this article focuses on solving dense SLEs but most LPs and MIPs are sparse, the REF factorization framework must be specialized to deal with sparsity before it can be effectively implemented within state-of-the-art exact LP and MIP solvers. These nontrivial steps will be undertaken in our future work.

2.2 Previous Related Computational Tests

Other methods for solving rational SLEs exactly (i.e., for validating basic solutions) have been previously studied. Cook and Steffy (2011) evaluated four prominent approaches within the context

of LP: exact rational arithmetic LU factorization, Wiedemann's black box algorithm (Kaltofen and Saunders 1991), p -adic lifting (see Dixon (1982)), and iterative refinement (see Wan (2006)). Their implementation of the latter two approaches followed largely the same blueprint as in Steffy (2011a) with significant improvements to the rational reconstruction subroutines they require to recover rational solutions from congruence-modulo and floating-point solutions, respectively. In all, iterative refinement had the best average performance, although the exhibited results exclude five instances where it failed due to numerical issues—conversely, the three other approaches were able to solve every instance to completion. The p -adic lifting approach followed close behind with runtimes that were approximately 15% slower. Despite its use of rational arithmetic, exact LU factorization was on average only three times slower than both of these methods. This performance is encouraging for the prospect of improved exact LU factorization methods since it was not far from the top two approaches, which benefitted from various enhancements listed in Chen and Storjohann (2005), Cook and Steffy (2011), and Steffy (2011a). On the other hand, appreciably more work would be required for Wiedemann's black-box algorithm to become competitive in this context since its runtimes were approximately 40 times slower than the top two algorithms.

Based on the takeaways from the thorough study conducted in Cook and Steffy (2011), exact LU factorization offers key advantages that could potentially render it among the top choices for performing basic solution validation within exact LP. In particular, p -adic lifting and iterative refinement are effective on simpler instances, but they become more costly/ineffective than the factorization approach when the problem is numerically unstable and/or when the solution size is not small. A problematic aspect of the latter statement is that solution size is generally indeterminable *a priori* and independent of the numerical condition of the basis matrix. Thus, it is conceivable that a seemingly ordinary right-hand side (RHS) vector could cause p -adic lifting and iterative refinement to achieve their worst-case performance for ill-conditioned and well-conditioned basis matrices alike. In light of these facts, rational arithmetic LU factorization is used as the point of reference for the computational tests performed in this article.

The most decisive argument for implementing exact LU factorization for basic solution validation is voiced by current practice of state-of-the-art exact LP solvers QSOpt_ex and SoPlex. Their most recent versions implement exact rational arithmetic LU factorization and not p -adic lifting or iterative refinement for basic solution validation. However, they also employ an advanced form of rational reconstruction known as output sensitive rational reconstruction (Steffy 2011a) (not shown in Figures 1(a) and (b)) to attempt to circumvent LU basis validation. Output-sensitive rational reconstruction is highly effective on simpler LP instances with small solution size but less successful on more intricate problems (Chen and Storjohann 2005; Cook and Steffy 2011; Steffy 2011a). For the 1,202 instances tested in Gleixner (2015) via SoPlex, the solver's pairing with output sensitive rational reconstruction was 5% faster than its pairing with LU basis validation only for instances that it solved in less than 0.5 seconds, whereas the latter implementation was 46% faster than the former over all instances. Nevertheless, we believe a future solver implementation could benefit from applying output sensitive rational reconstruction as a first measure of validation and, when it fails or its runtime exceeds 0.5 seconds, applying LU basis validation.

2.3 The REF Factorization Framework

The REF factorization framework is founded on the integer-preserving Gaussian Elimination algorithm (IPGE), which consists, as its names suggests, of integer-preserving row-reduction operations. A standard description of this algorithm can be found in Escobedo and Moreno-Centeno (2015). To proceed with the present discussion, it is necessary to introduce only a minimal description of IPGE as well as a small number of definitions and assumptions.

ASSUMPTION 1. Let $Ax = \mathbf{b}$ be a nonsingular system of linear equations, where $A \in \mathbb{Z}^{n \times n}$, $\mathbf{b} \in \mathbb{Z}^n$, and thus, $\mathbf{x} \in \mathbb{Q}^n$.

Definition 2.1. Let bit-length denote the units of computer storage for individual coefficients; for an integer, it is given simply by the number of bits its exact expression requires; for a rational, it is given by the sum of the bit-lengths of its integer numerator and its integer denominator.

Definition 2.2. Solution size is defined as the maximum bit-length required by an individual element of the solution vector.

Definition 2.3. Let A^k be the k th-iteration matrix of IPGE, for integer $0 \leq k \leq n$, where $A^0 := A$. Denote the individual entries of this matrix as $a_{i,j}^k$, for $1 \leq i, j \leq n$.

Definition 2.4. Let the scalar ρ^k denote the pivot element selected from A^{k-1} to perform the k th iteration of IPGE, where $\rho^0 := 1$.

Definition 2.5. The standard formula for calculating IPGE entry $a_{i,j}^k$ is as follows:

$$a_{i,j}^k = \begin{cases} a_{i,j}^{k-1} & \text{if } i = r_k \\ (\rho^k a_{i,j}^{k-1} - a_{r_k,j}^{k-1} a_{i,c_k}^{k-1}) / \rho^{k-1} & \text{otherwise} \end{cases} \quad \text{for } k = 1 \dots n, \quad (1)$$

where $1 \leq i, j, k \leq n$ and $\rho^k \neq 0$ for all k ; and where r_k and c_k are the row and column indices, respectively, of ρ^k in A^{k-1} .

ASSUMPTION 2. Fix $\rho^k = a_{k,k}^{k-1} \neq 0$ (i.e., $r_k = c_k = k$), for $k \geq 1$.

Assumption 2 implies IPGE does not need to perform row/column permutations between iterations to find nonzero pivot elements, and it is adopted for the purpose of simplicity. However, extended definitions of the algorithm (e.g., see Escobedo and Moreno-Centeno (2015)) incorporate row and column permutation matrices to deal with the possibility of encountering zeros along the diagonal during normal execution (for further clarification, the downloadable general purpose code associated with this article incorporates pivoting). It is important also to explain that IPGE and the REF factorization framework can be applied to solve rational SLEs by first performing simple transformations on the fixed-precision rational inputs—fractional entries are multiplied by their lowest common denominator and decimal entries are multiplied (i.e., right-shifted) by an adequate power of 10.

IPGE PROPERTY 1. Every division performed in the algorithm is guaranteed to be exact, that is, each dividend is an integer multiple of its divisor (Bareiss 1968).

IPGE PROPERTY 2. Each IPGE entry is equal to a specific subdeterminant of A (Edmonds 1967):

$$a_{i,j}^k = \begin{cases} (-1)^{i+k} \det(A)_{1 \dots i-1, i+1 \dots k, j}^{1 \dots k} & \text{if } i \leq k \\ \det(A)_{1 \dots k, j}^{1 \dots k, i} & \text{otherwise} \end{cases}, \quad (2)$$

for $0 \leq k \leq n$ and $1 \leq i, j \leq n$.

IPGE PROPERTY 3. Let $\sigma := \max_{i,j} |a_{i,j}^0|$. Then, the bit-length of an IPGE entry, denoted as $\beta_{\max}$, is bounded polynomially as $\beta_{\max} \leq \lceil n \log(\sigma \sqrt{n}) \rceil$ (Bareiss 1972).

IPGE serves as the foundation for the REF factorization framework (Escobedo and Moreno-Centeno 2015), which consists of algorithms for constructing the REF LU (REF-LU) and REF

Cholesky factorizations and performing REF triangular solves (forward and backward substitution). This article centers on the REF-LU, which is actually an $LD^{-1}U$ factorization whose composite nonzero entries equate with individual IPGE entries as follows:

$$l_{i,j} = a_{i,j}^{j-1}, \quad \text{for } i \geq j; \quad (3)$$

$$d_{i,i} = \rho^{i-1} \rho^i = a_{i-1,i-1}^{i-2} a_{i,i}^{i-1}, \quad \text{for all } i; \text{ and} \quad (4)$$

$$u_{i,j} = a_{i,j}^{i-1}, \quad \text{for } i \leq j; \quad (5)$$

where $1 \leq i, j \leq n$. Notice that D 's entries can be generated on the fly since $a_{i-1,i-1}^{i-2} a_{i,i}^{i-1} = l_{i-1,i-1} l_{i,i} = u_{i-1,i-1} u_{i,i}$, where $a_{0,0}^{-1} := \rho^0 = 1$, meaning D does not need to be stored. We remark that when A is symmetric, $U = L^T$, thereby inducing the REF Cholesky factorization $(LD^{-1/2})(LD^{-1/2})^T$; hence, the theory developed in this article encompasses both factorizations.

Recalling that $\mathbf{b} \in \mathbb{Z}^n$, REF forward substitution is performed by initializing and then iteratively updating vector $\mathbf{y} \in \mathbb{Z}^n$, for iterations $k = 0, \dots, n-1$, as follows:

$$y_i = \begin{cases} b_i & \text{if } k = 0, \\ l_{1,1}y_1 - l_{i,1}y_1 & \text{if } k = 1, \\ \frac{(l_{k,k}y_k - l_{i,k}y_k)}{l_{k-1,k-1}} & \text{otherwise} \end{cases} \quad ; \quad \text{for } i = k+1 \dots n. \quad (6)$$

The vector output at the completion of the above algorithm is the solution to the SLE $LD^{-1}\mathbf{y} = \mathbf{b}$. We note that Equation (6) is an equivalent but more memory-efficient version of the REF forward substitution algorithm given in Escobedo and Moreno-Centeno (2015). Specifically, this enhanced version writes over the elements of $\mathbf{y}$ at each iteration, whereas the original version stores an ancillary triangular array of $(n^2 + n)/2$ elements to output the same forward substituted vector.

It was shown in Escobedo and Moreno-Centeno (2015) that y_i equals IPGE entry $a_{i,n+1}^{i-1}$, where column index $n+1$ corresponds to the SLE's RHS (i.e., the vector $\mathbf{b}$ on which REF forward substitution is performed). To solve $A\mathbf{x} = \mathbf{b}$ without roundoff errors, it is important to point out that REF forward and backward substitution must be applied to the scaled SLE $A\mathbf{x}' = \mathbf{b}'$, where $\mathbf{x}' = \det(A)\mathbf{x}$ and $\mathbf{b}' = \det(A)\mathbf{b}$. This scaling guarantees $\mathbf{x}'$ is integral during backward substitution—which cannot be guaranteed for $\mathbf{x}$ —based on Cramer's rule. Nevertheless, the corresponding vector $\mathbf{y}'$ can be equivalently obtained without roundoff errors by evaluating Equation (6) and then multiplying $\mathbf{y}$ by $\det(A) = l_{n,n}$. Now, REF backward substitution is given by

$$x'_i = \frac{1}{u_{i,i}} \left(y'_i - \sum_{j=i+1}^n u_{i,j} x'_j \right) \quad \text{for } i = n \dots 1. \quad (7)$$

Following this process, the exact solution to the original SLE can be reported as

$$x_i = \frac{x'_i}{\det(A)} = \frac{x'_i}{l_{n,n}} \quad \text{for } i = 1 \dots n. \quad (8)$$

As a final remark, notice from Equations (6)–(8) that it is unnecessary to store D or D^{-1} —or to calculate any square roots in the case of the REF Cholesky factorization—when solving SLEs via REF forward and backward substitution, thus justifying the REF factorizations' names (Escobedo and Moreno-Centeno 2015).

3 FEATURED COMPUTATIONAL TESTS

The performance of the REF framework for solving dense SLEs is measured in relation to exact rational arithmetic LU factorization, which is a primary basic solution validation subroutine used in exact LP solvers (see Section 2.1). In addition, we devise and test an efficient adaptation of the Q-matrix revised simplex method (Azulay and Pique 2001), which is an alternative IPGE-based approach. The metrics of interest are threefold: (1) runtimes of the construction phases for each tool (i.e., calculating exact LU factorizations or adapted Q-matrices), (2) runtimes of the respective solution phases (i.e., performing exact forward and backward triangular solves or multiplying the adapted Q-matrix with the SLE RHS), and (3) storage requirements. Original derivations and other theoretical insights precede each set of computational tests.

3.1 Computing Environment

The experiments were performed on machines equipped with 22GB of RAM memory shared by two 2.8GHz quad core Intel Xeon 5560 processors. The code was written in C++ using the unlimited-precision GNU Multiple Precision Arithmetic Library (Granlund et al. 2015). Additionally, the code allows solutions to be output in user-specified precision floating point using the GNU Multiple Precision Floating-Point Reliable Library (Fousse et al. 2007).

3.2 General Instance Specifications

The following instance specifications are standard to all the featured computational tests, unless noted otherwise. The input basis or left-hand side (LHS) matrices of the linear systems are randomly generated fully dense matrices with dimensions $n \times n$. In the exact rational arithmetic LU factorization comparisons, the tested basis dimension parameters are $n = 50, 100, \dots, 500$; the upper limit was set to 500 due to the high runtimes required by rational arithmetic factorization in preliminary runs. In the Q-matrix comparisons, the tested basis dimension parameters are $n = 100, 200, \dots, 1000$. The input RHS matrices are randomly generated and have dimensions $n \times 50$, meaning the reported statistics apply to the sum of 50 distinct RHS vectors solved per repetition; this choice was made to obtain stable results since, for small n , the recorded runtimes for solving few RHS vectors can be insignificant or inconsistent. The input matrix entries are integers drawn uniformly from the interval $[-99, 99]$ (excluding zero). For each setting of the dimension parameter n , the experiments perform 30 repetitions and report the arithmetic mean and standard deviation of each metric of interest.

It is worthwhile to elaborate on the featured experimental setup, which was geared toward fully dense input matrices. During preliminary runs, we observed that the introduction of zeros along the working-matrix diagonal was rare due to the designated input matrix entry range and the explicit exclusion of zeros as inputs. Hence, pivot search subroutines were disabled in all computational tests and the respective pivot elements were selected from the next available diagonal element of the working matrix (i.e., according to Assumption 2)—we emphasize, however, that the code that has been made publicly available to accompany this work incorporates pivot searches, and is thus able to permute rows/columns accordingly. In the rare occurrence of a zero-pivot in the featured construction phase computational tests, the associated run was automatically discarded and replaced with a new instance (this happened in only 2 out of 780 random instances and was verified by running the experiments for a second time). A primary motivation for disabling pivot searches and standardizing pivot selection is to isolate the computational improvements associated with replacing rational arithmetic with integer arithmetic in the exact LU factorization process. Even so, because unlimited precision renders the numerical stability concern irrelevant, it is plausible that the exact rational arithmetic and REF factorizations would draw their pivots from the

same matrix coordinates when dealing with sparse matrices. This hypothesis is supported in part by the theoretical connections established in the next section, but its empirical verification falls outside of our present scope and is left for future work.

4 COMPARISON WITH EXACT RATIONAL ARITHMETIC LU FACTORIZATION

There are countless ways to factor a nonsingular matrix A into lower and upper triangular matrices L and U . However, requiring either L to be unit lower-triangular or U to be unit upper-triangular guarantees a unique factorization: the Doolittle LU factorization or the Crout LU factorization, respectively. These two elementary factorizations are constructed using distinct algorithms, whose descriptions can be found in Forsythe and Moler (1967) and Li et al. (2003), among others. The ensuing subsection establishes fundamental connections between the exact forms of both unique LU factorizations (hereafter denoted succinctly as Doolittle-LU and Crout-LU) and REF-LU. Then, Section 4.2 performs a series of computational experiments to compare the exact rational arithmetic and REF factorization frameworks for solving dense SLEs.

4.1 Connections between REF-LU and the Doolittle and Crout LU Factorizations

It is important to note that Doolittle-LU or Crout-LU—or any other LU factorization of A —is applied to solve the SLE $A\mathbf{x} = \mathbf{b}$ via the same procedure: a triangular solve of the (lower-triangular) SLE $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y} \in \mathbb{Q}^n$ followed by a triangular solve of the (upper-triangular) SLE $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x} \in \mathbb{Q}^n$. Both triangular solves must be done in exact rational arithmetic to avoid roundoff errors. Based on Cramer's rule, the outcome of this process can be expressed exactly as

$$\mathbf{x} = \frac{\text{Adj}(A)\mathbf{b}}{\det(A)}, \quad (9)$$

where $\text{Adj}(A)$ and $\det(A)$ are the adjunct matrix and determinant of A , respectively. A key related point is that each rational $x_i = [\text{Adj}(A)_{(i,:)}\mathbf{b}]/\det(A)$, for $1 \leq i \leq n$, can be simplified when its numerator and denominator share common factors—i.e., in such a situation the requisite bit-length of x_i is smaller than the summed bit-lengths of $\text{Adj}(A)_{(i,:)}\mathbf{b}$ and $\det(A)$. Then again, it is plausible that at least one of the solution numerators is relatively prime to $\det(A)$. This is noteworthy because, generally speaking, the bit-length of such a rational would be among the largest required by an individual entry of the solution vector. To inspect this issue further, an experiment described in Section 4.2.2 counts the number of tested instances in which at least one x_i has to store $\det(A)$ as its denominator (i.e., its exact rational expression cannot be simplified via GCD).

The following notational conventions are needed to establish the theoretical connections between REF-LU and Doolittle-LU and Crout-LU.

Definition 4.1. Let α_C^R be an abbreviation of $\det(A)_C^R$, the subdeterminant involving the rows and columns of A indexed by sets $R, C \subseteq \{1 \dots n\}$ such that $|R| = |C|$.

For the ensuing discussion, let $a_{i,j}^k$ denote the k th-iteration (i,j) -entry associated with applying IPGE to $A \in \mathbb{Z}^{n \times n}$ according to Assumption 2—which specifies that the k th IPGE pivot, ρ^k , is always given by $a_{k,k}^{k-1} \neq 0$ —for $1 \leq i, j, k \leq n$. In addition, fix $\bar{L}$, $\bar{D}^{-1}$, and $\bar{U}$ to be the three matrices comprising the REF LU factorization of A to differentiate them from the matrix factors of Doolittle-LU and Crout-LU.

THEOREM 4.2. *The Doolittle Factorization can be readily obtained from the REF LU factorization given by Equations (3)–(5).*

PROOF. The REF LU diagonal matrix $\bar{D}$ can be split into the diagonal matrices $\bar{D}^1, \bar{D}^2$ with nonzero entries given by $\bar{d}_{i,i}^1 = \rho^{i-1} = a_{i-1,i-1}^{i-2}$ and $\bar{d}_{i,i}^2 = \rho^i = a_{i,i}^{i-1}$ for all i , respectively. Now,

REF-LU can be re-factorized as follows:

$$\bar{L}\bar{D}^{-1}\bar{U} = \bar{L}(\bar{D}^1\bar{D}^2)^{-1}\bar{U} \quad (10)$$

$$= \bar{L}(\bar{D}^2)^{-1}(\bar{D}^1)^{-1}\bar{U} \quad (11)$$

$$= \begin{bmatrix} 1 & & & & & \\ \frac{a_{2,1}^0}{a_{1,1}^0} & 1 & & & & \\ \frac{a_{3,1}^0}{a_{1,1}^0} & \frac{a_{3,2}^1}{a_{2,2}^1} & 1 & & & \\ \vdots & \vdots & \ddots & \ddots & & \\ \frac{a_{n,1}^0}{a_{1,1}^0} & \frac{a_{n,2}^1}{a_{2,2}^1} & \dots & \frac{a_{n,n-1}^{n-2}}{a_{n-1,n-1}^{n-2}} & 1 & \end{bmatrix} \begin{bmatrix} a_{1,1}^0 & a_{1,2}^0 & a_{1,3}^0 & \dots & a_{1,n}^0 \\ & \frac{a_{2,2}^1}{a_{1,1}^0} & \frac{a_{2,3}^1}{a_{1,1}^0} & \dots & \frac{a_{2,n}^1}{a_{1,1}^0} \\ & \frac{a_{3,3}^2}{a_{2,2}^1} & \dots & \frac{a_{3,n}^2}{a_{2,2}^1} \\ & & \ddots & \ddots & \vdots \\ 0 & & & \frac{a_{n,n}^{n-1}}{a_{n-1,n-1}^{n-2}} \end{bmatrix}, \quad (12)$$

where Equation (12) right-multiplies $\bar{L}$ by $(\bar{D}^2)^{-1}$ and left-multiplies $\bar{U}$ by $(\bar{D}^1)^{-1}$. The result follows from the fact that there exists only one LU factorization with a unit lower-triangular matrix and, consequently, Equation (12) is equivalent to Doolittle-LU. $\square$

COROLLARY 4.3. *Doolittle-LU can be expressed via subdeterminants of A as follows:*

$$LU = \begin{bmatrix} 1 & & & & & \\ \frac{\alpha_1^2}{\alpha_1^1} & 1 & & & & \\ \frac{\alpha_1^3}{\alpha_1^1} & \frac{\alpha_{1,2}^{1,3}}{\alpha_{1,2}^{1,2}} & 1 & & & \\ \frac{\alpha_1^4}{\alpha_1^1} & \frac{\alpha_{1,2}^{1,4}}{\alpha_{1,2}^{1,2}} & \frac{\alpha_{1,2,3}^{1,2,4}}{\alpha_{1,2,3}^{1,2,3}} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \ddots & \\ \frac{\alpha_1^n}{\alpha_1^1} & \frac{\alpha_{1,2}^{1,n}}{\alpha_{1,2}^{1,2}} & \frac{\alpha_{1,2,3}^{1,2,n}}{\alpha_{1,2,3}^{1,2,3}} & \dots & \frac{\alpha_{1,\dots,n-1}^{1,\dots,n-2,n}}{\alpha_{1,\dots,n-1}^{1,\dots,n-1}} & 1 \end{bmatrix} \begin{bmatrix} \alpha_1^1 & \alpha_2^1 & \alpha_3^1 & \alpha_4^1 & \dots & \alpha_n^1 \\ & \frac{\alpha_{1,2}^{1,2}}{\alpha_1^1} & \frac{\alpha_{1,3}^{1,2}}{\alpha_1^1} & \frac{\alpha_{1,4}^{1,2}}{\alpha_1^1} & \dots & \frac{\alpha_{1,n}^{1,2}}{\alpha_1^1} \\ & \frac{\alpha_{1,2,3}^{1,2,3}}{\alpha_{1,2}^{1,2}} & \frac{\alpha_{1,2,4}^{1,2,3}}{\alpha_{1,2}^{1,2}} & \dots & \frac{\alpha_{1,2,n}^{1,2,3}}{\alpha_{1,2}^{1,2}} \\ & \frac{\alpha_{1,2,3,4}^{1,2,3,4}}{\alpha_{1,2,3}^{1,2,3}} & \dots & \frac{\alpha_{1,2,3,n}^{1,2,3,4}}{\alpha_{1,2,3}^{1,2,3}} \\ & & \ddots & \ddots & \vdots \\ 0 & & & \frac{\alpha_{1,\dots,n}^{1,\dots,n-1}}{\alpha_{1,\dots,n-1}^{1,\dots,n-1}} \end{bmatrix}. \quad (13)$$

To highlight the fundamental connection between Doolittle-LU and REF-LU, the contents of $\bar{L}$ and $\bar{U}$ can be alternatively expressed as subdeterminants of A as follows:

$$\bar{L} = \begin{bmatrix} \alpha_1^1 & & & & & \\ \alpha_1^2 & \alpha_{1,2}^{1,2} & & & & \\ \alpha_1^3 & \alpha_{1,2}^{1,3} & \alpha_{1,2,3}^{1,2,3} & & & \\ \alpha_1^4 & \alpha_{1,2}^{1,4} & \alpha_{1,2,3}^{1,2,4} & \alpha_{1,2,3,4}^{1,2,3,4} & & \\ \vdots & \vdots & \vdots & \ddots & \ddots & \\ \alpha_1^n & \alpha_{1,2}^{1,n} & \alpha_{1,2,3}^{1,2,n} & \dots & \alpha_{1,\dots,n-1}^{1,\dots,n-2,n} & \alpha_{1,\dots,n}^{1,\dots,n-1} \end{bmatrix}; \bar{U} = \begin{bmatrix} \alpha_1^1 & \alpha_2^1 & \alpha_3^1 & \alpha_4^1 & \dots & \alpha_n^1 \\ & \alpha_{1,2}^{1,2} & \alpha_{1,3}^{1,2} & \alpha_{1,4}^{1,2} & \dots & \alpha_{1,n}^{1,2} \\ & & \alpha_{1,2,3}^{1,2,3} & \alpha_{1,2,4}^{1,2,3} & \dots & \alpha_{1,2,n}^{1,2,3} \\ & & & \alpha_{1,2,3,4}^{1,2,3,4} & \dots & \alpha_{1,2,3,n}^{1,2,3,4} \\ & & & & \ddots & \vdots \\ 0 & & & & & \alpha_{1,\dots,n}^{1,\dots,n-1} \end{bmatrix}. \quad (14)$$

Notice that, except for the diagonal elements of each lower-triangular matrix, the numerators of L and U 's entries are exactly the corresponding entries of $\bar{L}$ and $\bar{U}$, and the denominators of L and U 's entries are different principal minors of A. These observations lead to the hypothesis that Doolittle-LU requires asymptotically twice the storage of REF-LU; this is because the fractional entries in Equation (15) consist of two subdeterminants that have the same order (or are off by one) and the expectation of relatively few common factors between each of these pairs. The ensuing theorem and corollary lead to the same conjecture with respect to Crout-LU.

THEOREM 4.4. *The Crout Factorization can be readily obtained from the REF LU factorization given by Equations (3)–(5).*

PROOF. The proof is similar to that of Theorem 4.2 and is omitted for brevity. $\square$

COROLLARY 4.5. *Crout-LU can be expressed via subdeterminants of A as follows:*

$$LU = \begin{bmatrix} \alpha_1^1 & & & & & & 0 \\ \alpha_1^2 & \frac{\alpha_{1,2}^{1,2}}{\alpha_1^1} & & & & & \\ \alpha_1^3 & \frac{\alpha_{1,2}^{1,3}}{\alpha_1^1} & \frac{\alpha_{1,2,3}^{1,2,3}}{\alpha_{1,2}^{1,2}} & & & & \\ \alpha_1^4 & \frac{\alpha_{1,2}^{1,4}}{\alpha_1^1} & \frac{\alpha_{1,2,3}^{1,2,4}}{\alpha_{1,2}^{1,2}} & \frac{\alpha_{1,2,3,4}^{1,2,3,4}}{\alpha_{1,2,3}^{1,2,3}} & & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & & \\ \alpha_1^n & \frac{\alpha_{1,2}^{1,n}}{\alpha_1^1} & \frac{\alpha_{1,2,3}^{1,2,n}}{\alpha_{1,2}^{1,2}} & \frac{\alpha_{1,2,3,4}^{1,2,3,n}}{\alpha_{1,2,3}^{1,2,3}} & \cdots & \frac{\alpha_{1,\dots,n}^{1,\dots,n}}{\alpha_{1,\dots,n-1}^{1,\dots,n-1}} \end{bmatrix} \begin{bmatrix} 1 & \frac{\alpha_2^1}{\alpha_1^1} & \frac{\alpha_3^1}{\alpha_1^1} & \frac{\alpha_4^1}{\alpha_1^1} & \cdots & \frac{\alpha_n^1}{\alpha_1^1} \\ & 1 & \frac{\alpha_{1,3}^{1,2}}{\alpha_{1,2}^{1,2}} & \frac{\alpha_{1,4}^{1,2}}{\alpha_{1,2}^{1,2}} & \cdots & \frac{\alpha_{1,n}^{1,2}}{\alpha_{1,2}^{1,2}} \\ & & 1 & \frac{\alpha_{1,4}^{1,2,3}}{\alpha_{1,2,3}^{1,2,3}} & \cdots & \frac{\alpha_{1,2,3,n}^{1,2,3}}{\alpha_{1,2,3}^{1,2,3}} \\ & & & 1 & \cdots & \frac{\alpha_{1,2,3,4}^{1,2,3,4}}{\alpha_{1,2,3,4}^{1,2,3,4}} \\ & & & & \ddots & \vdots \\ & & & & & 1 \end{bmatrix}. \quad (15)$$

4.2 Experiments and Results

This section compares the construction phases (i.e., the factorization subroutines) and the solution phases (i.e., the respective triangular solves) for REF-LU, Doolittle-LU, and Crout-LU computationally. The standard arithmetic functions of the GMP library were employed to code each of the algorithms. However, in the case of the REF factorization and solution algorithms, we used Jebelean's exact division algorithm (Jebelean 1997) (also offered within GMP) to take advantage of IPGE Property 1. Jebelean's division algorithm halves the work of the default division algorithm (Granlund et al. 2015), and it can be employed by the REF algorithms because their constituent divisions are guaranteed to be exact.

4.2.1 Runtimes. The first experiment measures the runtimes, in seconds (s), required by the construction and solution phases of the three exact LU factorizations. Table 1 summarizes factorization runtimes for Crout-LU (label **Crout**), Doolittle-LU (label **Doolittle**), and REF-LU (label **REF**) via average (AVG) and standard deviation (SD) statistics. Table 2 summarizes the respective triangular solve runtimes; the reported statistics apply to the sum of 50 RHS vectors solved per each of 30 repetitions. Both tables also summarize the ratios of the runtimes taken by each rational arithmetic LU factorization to the runtimes taken by the REF LU factorization (labels **Crout/REF** and **Dool/REF**).

As Tables 1 and 2 demonstrate, REF factorization significantly outperformed exact rational arithmetic factorization. REF-LU construction phase average runtimes were up to 22 and 17 times faster than those achieved by Crout-LU and Doolittle-LU, respectively. Its relative performance reached a peak around $n = 250$, and then gradually decreased as n increased; this is likely due to the growing influence of slower-memory operations affecting both approaches as the associated matrix entry bit-lengths become larger. Nonetheless, REF factorization was still able to handle a fully dense 500×500 matrix—i.e., with 250,000 nonzeros—in under a minute, while each exact rational arithmetic factorization took over 15 minutes. Even more impressively, REF-LU solution phase average runtimes were over 35 times faster than both Crout-LU and Doolittle-LU's across the board. The reported solution runtimes for $n = 500$ translate to roughly .5s per RHS vector for REF-LU and 19s for exact rational arithmetic factorization since 50 RHS vectors were solved per repetition.

4.2.2 Storage. The second experiment performs a new set of runs to examine the memory requirements of REF-LU, Doolittle-LU, and Crout-LU. To this end, it measures each factorization's

Table 1. Construction Phase (Factorization) Runtimes (s)

n	Crout		Doolittle		REF		Crout/REF		Dool/REF	
	AVG	SD	AVG	SD	AVG	SD	AVG	SD	AVG	SD
50	0.06	0.01	0.05	0.00	0.01	0.00	11.64	1.96	9.12	0.35
100	1.14	0.02	0.89	0.01	0.07	0.00	17.25	0.44	13.46	0.29
150	6.46	0.03	5.03	0.03	0.31	0.00	20.97	0.17	16.32	0.14
200	21.90	0.08	17.19	0.07	0.99	0.00	22.15	0.11	17.39	0.10
250	56.35	0.16	44.89	0.21	2.55	0.01	22.06	0.12	17.57	0.10
300	122.03	0.28	98.85	0.34	5.85	0.07	20.85	0.25	16.89	0.20
350	234.55	0.48	193.18	0.60	11.94	0.32	19.66	0.48	16.19	0.39
400	414.32	0.85	346.57	0.83	21.79	0.17	19.02	0.15	15.91	0.13
450	684.03	0.96	581.63	1.68	37.06	0.45	18.46	0.21	15.70	0.19
500	1073.25	1.49	924.51	3.15	59.67	0.37	17.99	0.11	15.49	0.09

Table 2. Solution Phase Runtimes (s) (Statistics Apply to theSum of 50 RHS Vectors Solved per Repetition)

n	Crout		Doolittle		REF		Crout/REF		Dool/REF	
	AVG	SD	AVG	SD	AVG	SD	AVG	SD	AVG	SD
50	0.50	0.01	0.49	0.00	0.01	0.00	35.55	0.63	35.12	0.23
100	4.84	0.01	4.82	0.03	0.09	0.00	52.35	0.29	52.14	0.49
150	18.04	0.07	17.97	0.06	0.32	0.01	55.67	1.14	55.46	1.11
200	45.86	0.15	45.72	0.14	0.86	0.00	53.49	0.28	53.32	0.25
250	94.80	0.24	94.68	0.18	1.92	0.01	49.39	0.27	49.33	0.27
300	172.11	0.34	172.07	0.31	3.84	0.02	44.86	0.24	44.85	0.23
350	285.75	0.51	285.51	0.55	6.72	0.04	42.50	0.26	42.47	0.28
400	444.13	0.67	444.33	1.62	10.84	0.06	40.97	0.24	40.99	0.31
450	656.31	0.98	656.26	1.10	16.62	0.14	39.48	0.32	39.48	0.34
500	933.21	1.59	932.84	1.52	24.41	0.25	38.23	0.38	38.22	0.38

required number of *limbs*, which is a system-dependent unit of storage corresponding to the part of a multi-precision number that fits in a single machine word (Granlund et al. 2015); in our system, it is a 64-bit long but in others it may be a 32-bit int or a long long. The construction storage requirements are summarized in Table 3a in terms of the average number of limbs utilized by REF-LU, rounded to the nearest whole number, for each specified matrix dimension n ; the standard deviation magnitudes were at most 0.5% of each average value and are omitted for visual clarity. The average number of limbs utilized by Crout-LU and Doolittle-LU is expressed as a ratio of each corresponding REF-LU average value rounded to four decimal places. For these runs, only REF-LU is explicitly constructed, while the exact rational arithmetic LU factorizations are obtained through the shortcuts implied by the findings of Section 4.1. That is, a denominator equal to a principal minor of A (i.e., an IPGE pivot) is appended to each REF-LU entry as depicted in Equations (13) and (15), and the resulting fractions are simplified via GCD operations. As an added benefit, this shortcut procedure makes it straightforward to determine the factors that are removed from an original REF-LU entry to yield the smallest possible numerator in the corresponding entries of Crout-LU and Doolittle-LU; this conforms to what is done by the GMP library, which stores each

Table 3. Storage Metrics

(a) Utilized Limbs				(b) Max Removed Factors		
n	REF-LU AVG	Crout/REF AVG	Dool/REF AVG	n	Crout MAX	Doolittle MAX
50	6235	1.9583	1.9574	50	957783288	87071208
100	46571	1.9757	1.9753	100	5853540	2862990
150	155777	1.9822	1.9823	150	3357228	23500596
200	370251	1.9862	1.9862	200	112150896	12628616
250	727506	1.9890	1.9891	250	2810372084	6433882
300	1265193	1.9908	1.9909	300	26199300	55207425
350	2022175	1.9920	1.9921	350	16536240	25652700
400	3038136	1.9929	1.9929	400	270198825	56812032
450	4351434	1.9937	1.9937	450	7801242	20072132
500	6002661	1.9944	1.9944	500	34010361	121091232

rational entry in canonical form (i.e., its numerator and denominator are relatively prime). The removed factors are of interest because they can be regarded as the memory cost of not having to store individual denominators for REF-LU's entries. Table 3b summarizes these costs, for each value of n , by recording the maximum such integer that is factored out of any REF-LU entry over the full corresponding set of repetitions.

The computational results confirm the hypothesized storage relationship between REF-LU and exact rational arithmetic factorization. Indeed, Table 3a shows that Crout-LU and Doolittle-LU utilized twice the number of average limbs as REF-LU asymptotically. Table 3b reveals that the maximum factor removed from a REF-LU entry to transform it into the corresponding denominator of Crout-LU or Doolittle-LU was at most 10 digits long over the full set of runs. Since a long can store 19 digits, this means in all of these runs the numerators of the exact rational arithmetic LU factorizations alone utilized the same number of limbs as the full REF LU factorization. Thus, the cost paid by REF-LU to avoid storing matrix-entry denominators was insignificant compared with the substantial storage savings from said avoidance.

The second experiment also records three metrics related to the solutions obtained via the exact rational arithmetic approach (label ERA) and the REF framework: (1) solution size—i.e., the maximum bit-length of an individual entry in the solution vector—in limbs; (2) determinant bit-length in limbs; (3) number of instances in which the determinant must be explicitly stored as a denominator for at least one entry in the exact rational arithmetic solution vector. Table 4 summarizes the results of these three aforesaid metrics. For the purpose of uniformity, the SLE $Ax = e_1$ is solved for all of these tests, where e_1 is the first n -length elementary vector. Hence, the RHS vector remains constant, whereas the LHS A matrices are again different randomly generated dense matrices. Each of these SLEs is a representative choice since it corresponds to solving for the first column of the inverse matrix; similar SLEs were solved in Steffy (2011b). Here, it was not necessary to differentiate between Doolittle-LU and Crout-LU results since the respective outputs of their triangular solves are the same, namely $x \in \mathbb{Q}^n$. However, it is important to emphasize that the solution $x' \in \mathbb{Z}^n$ returned by the REF framework stores only the numerator in Cramer's rule, given by Equation (8). This means x would be ultimately reported by appending $\det(A)$ as the common denominator for every element of x' (see Equation (8)). Conversely, the denominators of an ERA solution may differ from one another due to recurring cancellations of common factors (i.e., GCD operations)

Table 4. Solution Metrics

(a) Solution Size							(b) Determinant Statistics			
n	ERA Limbs		REF Limbs		ERA/REF		n	Limbs		Determinant= ERA Denom
	AVG	SD	AVG	SD	AVG	SD		AVG	SD	
50	13.97	0.18	7.00	0.00	2.00	0.03	50	7.00	0.00	10 of 30 instances
100	27.87	0.35	13.97	0.18	2.00	0.01	100	14.00	0.00	8 of 30 instances
150	42.00	0.00	21.00	0.00	2.00	0.00	150	21.00	0.00	11 of 30 instances
200	56.03	0.18	28.00	0.00	2.00	0.01	200	28.13	0.35	13 of 30 instances
250	72.00	0.00	36.00	0.00	2.00	0.00	250	36.00	0.00	13 of 30 instances
300	88.00	0.00	44.00	0.00	2.00	0.00	300	44.00	0.00	17 of 30 instances
350	103.33	0.55	51.50	0.51	2.00	0.00	350	52.00	0.00	11 of 30 instances
400	118.63	0.49	59.00	0.00	2.00	0.01	400	59.73	0.45	15 of 30 instances
450	134.33	0.48	67.00	0.00	2.00	0.01	450	67.40	0.50	15 of 30 instances
500	150.60	0.50	75.00	0.00	2.00	0.01	500	75.73	0.45	11 of 30 instances

with individual numerators, meaning n distinct denominators must be explicitly stored throughout the execution of exact rational arithmetic forward and backward substitution.

Table 4a evinces a significant $2\times$ solution size advantage for REF-LU arising from the fact that the output of REF backward substitution $\mathbf{x}'$ is implicitly accompanied by $\det(A)$ as the denominator of each of its n elements. As it happens, this implicit common denominator comes gratis to the REF framework since the last diagonal element of REF-LU's lower-triangular (or upper-triangular) matrix is equal to $\det(A)$. This is significant because on average $\det(A)$ had practically the same length as the REF-LU solution size, as Table 4b reports; by contrast, the exact rational arithmetic approach must explicitly store separate denominators that may each be as large as $\det(A)$, as the preceding paragraph explains. In point of fact Table 4b shows that for each tested value of n , the ERA solution required the determinant to be explicitly stored as a denominator in at least 26% of the instances—i.e., at least one numerator in the exact solution was relatively prime to $\det(A)$. Since the largest REF-LU bit-length usually corresponds to the basis determinant, this suggests that the REF factorization framework routinely stores the exact solution to $A\mathbf{x} = \mathbf{b}$ as compactly as possible.

5 COMPARISON WITH EDMOND'S Q-MATRIX APPROACH

Edmonds and Maurras (1997) devised an integer-preserving tool to solve rational SLEs exactly, which has come to be popularly known as Edmonds's Q-matrix approach. Each Q-matrix corresponds to the adjunct matrix of a basis and, like the REF factorizations, it is constructed via IPGE. Azulay and Pique (2001) expanded this tool by developing the Q-matrix revised simplex method, which was designed to operate as a full-fledged exact LP solver. In the limited accompanying set of experiments, their method decreased the respective exact rational arithmetic revised simplex run-times by at most one order of magnitude—achieving a maximum speedup of 12 on 1 of 24 NETLIB instances. Nevertheless, even if their method could sustain this performance edge, the savings would not suffice to overturn the conclusion reached in several implementations of exact rational arithmetic LP solvers (e.g., Cook and Steffy (2011)), namely that deploying an unlimited-precision LP solver is impractical in most situations. For this reason, the next subsection first derives an efficient adaptation of Q-matrices that enables their implementation as a basic solution validation

subroutine. Afterward, Section 5.2 compares the REF factorization framework to this streamlined Q-matrix approach.

5.1 A Streamlined Q-Matrix Approach for Basic Solution Validation

To validate a basic solution, it is unnecessary to perform individual operations within the simplex algorithm. In this context, the only essential parts of the Q-matrix revised simplex method are the construction of the adjunct matrix from a predetermined set of column indices and its implementation in calculating the exact solution to the underlying SLE. Assuming A is the basis matrix to be validated, the first part can be accomplished in $O(n^3)$ operations by applying the full version of IPGE to the augmented matrix $[A|I_n]$, so that its LHS becomes a diagonal matrix with all nonzero entries equal to $\det(A)$; $\text{Adj}(A)$ emerges as the accompanying right-hand matrix based on Cramer's rule. After this initial construction, the second essential part of the Q-matrix method is easily performed in $O(n^2)$ operations by a matrix-vector multiplication involving $\text{Adj}(A)$ and a RHS vector $\mathbf{b}$ (in the case of the primal solution calculation).

Based on the preceding discussion, it is necessary to allocate an $n \times 2n$ matrix to construct $\text{Adj}(A)$, but only an $n \times n$ matrix during the exact solution calculation. Next, we introduce an alternative construction phase algorithm requiring an $n \times n$ matrix, thereby reducing the previous storage requirement by one-half. The algorithm is described by the following equation, which needs to be evaluated for $k = 1 \dots n$, $i = 1 \dots n$, and $j = 1 \dots k-1, k+1 \dots n, k$ (in this specific order):

$$a_{i,j}^k = \begin{cases} \rho^{k-1} & \text{if } i = k, j = k \\ -a_{i,j}^{k-1} & \text{if } i \neq k, j = k \\ a_{i,j}^{k-1} & \text{if } i = k, j \neq k \\ (\rho^k a_{i,j}^{k-1} - a_{k,j}^{k-1} a_{i,k}^{k-1}) / \rho^{k-1} & \text{if } i \neq k, j \neq k \end{cases} \quad (16)$$

where $a_{i,j}^0 = a_{i,j}$. The algorithm is identical to the full version of IPGE specified by Equation (1), except that the k th pivot column, $A_{(:,k)}^{k-1}$, is overwritten by itself times -1 in rows $i \neq k$ and by the previous pivot, ρ^{k-1} , in row k . It is straightforward to deduce that the algorithm performs $O(n^3)$ operations and that it requires storing only an $n \times n$ matrix. In what follows, Lemma 5.1 will provide the rationale for overwriting the pivot column at each step with the values indicated in Equation (16); afterward, Theorem 5.2 proves that the above algorithm is correct and REF. To this end, let $A_{(:,n+j)}^k$ denote the k th-iteration j th RHS column obtained when applying IPGE—according to the sequence of pivot coordinates designated by Assumption 2—on $[A|I_n]$, where $1 \leq j, k \leq n$.

LEMMA 5.1. *Provided that the $(k-1)$ th-iteration RHS column, $A_{(:,n+j)}^{k-1}$, equals $\rho^{k-1} \mathbf{e}_j$, for $k > 0$, the next iterate of the same column, $A_{(:,n+j)}^k$, can be obtained via the following shortcuts:*

$$A_{(:,n+j)}^k = \begin{cases} -A_{(:,k)}^{k-1} + (\rho^k + \rho^{k-1}) \mathbf{e}_k & \text{if } j = k \\ \rho^k \mathbf{e}_j & \text{otherwise} \end{cases},$$

where $\mathbf{e}_j$ is the n -length j th elementary vector, ρ^k is the k th IPGE pivot element and $\rho^0 = 1$.

PROOF. The proof is divided into two parts. When $j = k$, applying the definition of IPGE to the calculation of the individual entries of $A_{(:,n+j)}^k$ gives

$$a_{i,n+k}^k = \begin{cases} a_{i,n+k}^{k-1} = a_{k,n+k}^{k-1} & = \rho^{k-1} \quad \text{if } i = k \\ (\rho^k a_{i,n+k}^{k-1} - a_{k,n+k}^{k-1} a_{i,k}^{k-1}) / \rho^{k-1} = (\rho^k(0) - \rho^{k-1} a_{i,k}^{k-1}) / \rho^{k-1} = -a_{i,k}^{k-1} & \text{otherwise} \end{cases},$$

where ρ^{k-1} is substituted in place of $a_{k,n+k}^{k-1}$ according to the given assumption. Notice that, since $a_{k,k}^{k-1} = \rho^k$, the case $i = k$ can be equivalently expressed as

$$a_{k,n+k}^k = -a_{k,k}^{k-1} + (a_{k,k}^{k-1} + \rho^{k-1}) = -a_{k,k}^{k-1} + (\rho^k + \rho^{k-1}).$$

Similarly, for $j \neq k$, starting with the IPGE definition yields:

$$a_{i,n+j}^k = \begin{cases} a_{i,n+j}^{k-1} & = 0 & \text{if } i = k \\ (\rho^k a_{i,n+j}^{k-1} - a_{k,n+j}^{k-1} a_{i,k}^{k-1}) / \rho^{k-1} = (\rho^k \rho^{k-1} - (0) a_{i,k}^{k-1}) / \rho^{k-1} = \rho^k & \text{if } i = j \\ (\rho^k a_{i,n+j}^{k-1} - a_{k,n+j}^{k-1} a_{i,k}^{k-1}) / \rho^{k-1} = (\rho^k (0) - (0) a_{i,k}^{k-1}) / \rho^{k-1} = 0 & \text{otherwise} \end{cases} \quad \square$$

THEOREM 5.2. *The algorithm specified by Equation (16) yields $\text{Adj}(A)$ without accruing roundoff errors.*

PROOF. Since the iterative LHS matrix obtained by applying IPGE on $[A|I_n]$ is only needed to transform I_n into $\text{Adj}(A)$, columns $A_{(:,1)}^{k-1}, \dots, A_{(:,k-1)}^{k-1}$ are superfluous during the algorithm's k th to n th iterations. Hence, after completing the k th iteration, a new column can be inserted in place of $A_{(:,k)}^k$ without losing any critical data. The following analysis explains that the inserted column corresponds to $A_{(:,n+k)}^k$ and that only the first k RHS columns need to be explicitly stored up to this point.

Continuing, since $\rho^0 = 1$, the n -order identity matrix can be characterized as

$$I_n = \rho^0 I_n = [\rho^0 \mathbf{e}_1 \quad \rho^0 \mathbf{e}_2 \quad \dots \quad \rho^0 \mathbf{e}_n].$$

This implies that Lemma 5.1 can be applied in the first IPGE iteration on all the RHS columns of $[A^0|I_n]$ to obtain $A_{(:,n+1)}^1, \dots, A_{(:,2n)}^1$ via its shortcuts. In the second IPGE iteration, Lemma 5.1 will apply to all but the first of these columns since $A_{(:,n+2)}^1 = \rho^1 \mathbf{e}_2, \dots, A_{(:,2n)}^1 = \rho^1 \mathbf{e}_n$, but $A_{(:,n+1)}^1 = -A_{(:,n+1)}^0 + (\rho^1 + \rho^0) \mathbf{e}_1 \neq \rho^1 \mathbf{e}_1$. Propagating this simple argument, in iteration k the shortcuts can be applied only to columns $A_{(:,n+k)}^{k-1} = \rho^{k-1} \mathbf{e}_k, \dots, A_{(:,2n)}^{k-1} = \rho^{k-1} \mathbf{e}_n$. More importantly, since the content of these columns is readily deducible according to Lemma 5.1, only the first k RHS columns need to be explicitly stored after this iteration. As it happens, they can occupy exactly the first k columns of the LHS matrix since the columns $A_{(:,1)}^k, \dots, A_{(:,k)}^k$ are superfluous at this point, as the above paragraph explained. Thus, by first performing the usual IPGE operations on the k th-iteration non-pivot columns and then generating column $A_{(:,n+k)}^k$ to replace the pivot column $A_{(:,k)}^{k-1}$ via Lemma 5.1, for $k = 1$ to n , the algorithm will yield $\text{Adj}(A)$ as the output LHS matrix. In other words, there is no need to store the RHS matrix at any point. Lastly, since this algorithm is simply a storage-efficient version of the normal application of IPGE on $[A|I_n]$, all its calculations are free of roundoff errors. $\square$

The Q-matrix revised simplex method can now be streamlined for basic solution validation and compared in a leveled field with the REF factorization framework. Expressly, like REF-LU, $\text{Adj}(A)$ can be computed in $O(n^3)$ operations without roundoff errors using only an $n \times n$ matrix in the construction phase, after which it can be deployed to calculate the exact solution to $A\mathbf{x} = \mathbf{b}$ in $O(n^2)$ operations. In this case, however, the solution phase (for obtaining the exact primal solution) consists of the following matrix-vector multiplication:

$$\mathbf{x}' = \text{Adj}(A)\mathbf{b} = [\det(A)A^{-1}]\mathbf{b} = \det(A)\mathbf{x},$$

where the second equality comes from the definition of $\text{Adj}(A)$ and the third from the algebraic expression for $\mathbf{x}$ as a solution to $A\mathbf{x} = \mathbf{b}$. Notice that the multiplication yields the same scaled vector

$\mathbf{x}'$ as the result of REF forward and backward substitution and, thus, the exact solution to $\mathbf{Ax} = \mathbf{b}$ would be returned as

$$x_i = \frac{x'_i}{\det(A)} \quad \text{for } i = 1 \dots n.$$

Since the adjunct matrix is defined as the transpose cofactor matrix, $\text{Adj}(A)$ can be expressed in terms of subdeterminants of A (compare with Equations (13)–(15)) as follows:

$$\text{Adj}(A) = \begin{bmatrix} \alpha_{N \setminus \{1\}}^{N \setminus \{1\}} & -\alpha_{N \setminus \{1\}}^{N \setminus \{2\}} & \alpha_{N \setminus \{1\}}^{N \setminus \{3\}} & \dots & -\alpha_{N \setminus \{1\}}^{N \setminus \{n\}} \\ -\alpha_{N \setminus \{2\}}^{N \setminus \{1\}} & \alpha_{N \setminus \{2\}}^{N \setminus \{2\}} & -\alpha_{N \setminus \{2\}}^{N \setminus \{3\}} & \dots & \alpha_{N \setminus \{2\}}^{N \setminus \{n\}} \\ \alpha_{N \setminus \{3\}}^{N \setminus \{1\}} & -\alpha_{N \setminus \{3\}}^{N \setminus \{2\}} & \alpha_{N \setminus \{3\}}^{N \setminus \{3\}} & \dots & -\alpha_{N \setminus \{3\}}^{N \setminus \{n\}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\alpha_{N \setminus \{n\}}^{N \setminus \{1\}} & \alpha_{N \setminus \{n\}}^{N \setminus \{2\}} & -\alpha_{N \setminus \{n\}}^{N \setminus \{3\}} & \dots & \alpha_{N \setminus \{n\}}^{N \setminus \{n\}} \end{bmatrix}, \quad (17)$$

where $N = \{1, \dots, n\}$ is the column index set and n is even (solely for the purposes of this illustration). Based on Hadamard's determinant inequality, the bit-length of *every* entry of $\text{Adj}(A)$ has the bound $(n-1) \log(\sqrt{(n-1)\sigma})$; on the other hand, only the (n, n) -entry of REF-LU has this bit-length bound. The entries along row i of U and column i of L in REF-LU have bit-lengths with the bound $i \log(\sqrt{i\sigma})$ since they are equivalent to lower-order determinants (i.e., see the expression for $\bar{L}$ and $\bar{U}$ given by Equation (14)). This suggests that REF-LU requires significantly less storage than $\text{Adj}(A)$.

5.2 Experiments and Results

The remainder of this section compares the REF factorization framework with the streamlined Q-matrix approach computationally. The two main experiments developed for this purpose are analogous to those described in Sections 4.2.1 and 4.2.2.

5.2.1 Runtimes. The first experiment measures the runtimes of the respective construction and solution phases. The former consist of computing the REF LU factorization and $\text{Adj}(A)$ via the streamlined Q-matrix approach (hereafter alternatively referred to as Q-Mat-Adj); the respective solution phases consist of the REF triangular solves and a matrix-vector multiplication (see Section 5.1). For this experiment, the tested basis dimension parameters are $n = 100, 200, \dots, 1000$, meaning the largest input matrices tested have exactly 1,000,000 nonzeros. Table 5 summarizes the respective construction phase runtimes of each approach separately and as the ratio of Q-Mat-Adj to REF-LU (label Q-Mat/REF). Table 6 provides a summary of the solution phase runtimes; the reported statistics apply to the sum of 50 RHS vectors solved per each of 30 repetitions. For this metric, the table reports the ratios of REF-LU to Q-Mat-Adj (label REF/Q-Mat).

Notice that REF-LU once again achieved the best construction phase runtimes, this time outperforming Q-Matrix-Adj by close to a factor of nine. Moreover, it appears this performance edge would continue its upward trend if higher values of n were tested. Q-Matrix-Adj manages to perform twice as fast as exact rational arithmetic factorization, which is attested by cross-referencing Tables 1 and 5 along their overlapping n values. Most notably, Q-Mat-Adj beats REF-LU in terms of solution phase runtimes by upwards of a factor of 22, which also appears to rise for higher values of n than those tested. This reversal in relative performance is to be expected based on the different subroutines involved. Q-Mat-Adj simply computes an integer matrix-vector multiplication, while REF-LU performs REF forward and backward substitution, both of which include carrying out exact integer divisions.

Table 5. Construction Phase Runtimes (s)

n	Q-Mat-Adj		REF-LU		Q-Mat/REF	
	AVG	SD	AVG	SD	AVG	SD
100	0.33	0.02	0.07	0.00	5.05	0.20
200	7.27	0.03	0.99	0.04	7.33	0.23
300	45.50	0.21	5.86	0.05	7.76	0.06
400	172.76	0.38	21.91	0.14	7.88	0.04
500	486.61	2.49	60.53	0.32	8.04	0.04
600	1143.49	3.95	139.75	1.12	8.18	0.06
700	2367.04	12.10	283.59	1.52	8.35	0.05
800	4468.37	16.56	525.30	2.50	8.51	0.04
900	7933.58	82.07	916.87	13.41	8.65	0.10
1000	13279.76	107.65	1508.10	22.52	8.81	0.12

Table 6. Solution Runtimes (s) (Statistics Apply to the Sum of 50 RHS Vectors Solved per Repetition)

n	Q-Mat-Adj		REF-LU		REF/Q-Mat	
	AVG	SD	AVG	SD	AVG	SD
100	0.03	0.00	0.09	0.00	2.71	0.04
200	0.18	0.00	0.86	0.03	4.72	0.19
300	0.49	0.01	3.86	0.04	7.84	0.24
400	1.13	0.01	10.83	0.04	9.56	0.11
500	2.08	0.01	24.43	0.20	11.76	0.10
600	3.34	0.04	47.61	0.37	14.24	0.17
700	5.06	0.02	83.23	0.41	16.46	0.09
800	7.34	0.11	135.67	0.65	18.49	0.27
900	10.46	0.27	211.73	2.44	20.25	0.46
1000	14.15	0.25	313.99	3.53	22.20	0.41

5.2.2 Storage. The second experiment examines the respective memory requirements. REF-LU's requirements are summarized in Table 7a using average number of limbs, rounded to the nearest whole number, for each tested n value (the standard deviation magnitudes are omitted due to their relatively insignificant magnitude). Requirements for constructing Q-Mat-Adj are expressed as average ratios of the corresponding REF-LU average values (label Q-Mat/REF) rounded to four decimal places. As Section 5.1 explains, the solution phase subroutines of both methods yield the scaled solution vector $\mathbf{x}' = \text{Adj}(A)\mathbf{b}$ associated with IPGE. Hence, the IPGE solution size—i.e., the maximum bit-length of an element of $\mathbf{x}'$ —displayed in Table 7b applies to both approaches; the table reports the average solution sizes and their standard deviations in terms of bit-length and limbs. The RHS vector is fixed as $\mathbf{b} = \mathbf{e}_1$ as in Section 4.2.2.

Table 7a shows that Q-Mat-Adj requires approximately three times the storage of REF-LU and that this value will continue to rise gradually with n . It is reasonable to expect that this advantage would be even more pronounced for sparse matrices since LU factorizations can be designed to minimize fill-in while the inverse matrix is generally dense (Forsythe and Moler 1967; Murtagh 1981; Gilbert 1994). Q-Mat-Adj actually requires over 50% more storage than exact rational arithmetic LU factorization, as evinced by the results of Section 4.2.2. Additional tests in Section 5.2.3

Table 7. Storage Metrics

(a) Utilized Limbs			IPGE Solution Size			
n	REF-LU	Q-Mat/REF	Bit-length		Limbs	
	AVG	AVG	AVG	SD	AVG	SD
100	46585	2.9610	837.27	2.89	13.93	0.25
200	370236	3.0251	1782.37	1.61	28.00	0.00
300	1265339	3.1292	2764.03	2.62	44.00	0.00
400	3037594	3.1077	3770.13	2.42	59.00	0.00
500	6003518	3.1232	4795.43	2.71	75.03	0.18
600	10482385	3.1593	5835.30	2.59	92.00	0.00
700	16797440	3.1505	6887.43	2.24	108.00	0.00
800	25280548	3.1645	7948.97	1.96	125.00	0.00
900	36262807	3.1496	9020.70	2.45	141.07	0.25
1000	50076523	3.1552	10100.07	2.82	158.00	0.00

further examine how these elevated memory requirements impact the relative computational performances of all three featured approaches as n increases.

By stating solution size in terms of bit-length, the results of Table 7b can be related to the computational tests performed in Cook and Steffy (2011). In said study, solution size is used as one of three attributes for grouping instances. It is shown therein that when solution size is large—characterized as exceeding 1000 bits—the average performance of the exact rational arithmetic LU factorization approach is better than that of p -adic lifting and Wiedemann’s method; additionally, it is reasonably close to the top average performance achieved by iterative refinement. Since the exact rational arithmetic solution vector tends to require nearly twice the IPGE solution size in limbs (see Table 4a), this implies the solution of $Ax = e_1$ for all these instances yields “large” rational solution sizes. To verify this claim, we divided each IPGE solution vector x' by $\det(A)$ to yield x , and then reduced each resulting fraction to canonical form and measured its bit-length. Interestingly, the average size of x was 2.00 times that of x' for each value of n . Hence, nearly all problems tested in our experiments are likely to belong to the class of instances where the exact rational arithmetic factorization approach achieves a better performance than p -adic lifting and Wiedemann’s method and a slightly worse performance than iterative refinement, according to the aforementioned study. It is important to mention that the basis determinant, which implicitly accompanies the IPGE solution to obtain x from x' , is available as the last diagonal element of REF-LU but is not present in Q-Mat-Adj (it is obtained during its construction, but must be stored separately). Juxtaposed with the results of Section 4.2.2, this strongly suggests that the REF framework stores the exact solutions to dense SLEs more efficiently than the Q-matrix and exact rational arithmetic LU factorization approaches.

5.2.3 Extended Computational Tests. Tables 8 and 9 summarize the results of extended computational tests on Doolittle-LU, Q-Mat-Adj, and REF-LU. The featured tests replicate those done in Section 4.2.1 and Section 5.2.1 with the distinction that the basis dimension parameter n is chosen from powers of 2 rather than from multiples of 50. Specifically, the tested dimensions are $n = 2, 4, \dots, 2048$ for REF-LU and $n = 2, 4, \dots, 1024$ for Doolittle-LU and Q-Mat-Adj; the $n = 2048$ setting—i.e., the input matrix with exactly 4,194,304 nonzeros—was not tested for Doolittle-LU or Q-Mat-Adj since their construction phases (without which the solution phases cannot be performed) are projected to take approximately 1 week for each instance. On a related note, due to the longer runtimes expected in this experiment, the corresponding repetitions were performed

Table 8. Extended Tests of Construction Phase Runtimes (s)

n	Doolittle-LU		Q-Mat-Adj		REF-LU		Dool/REF	Q-Mat/REF
	AVG	SD	AVG	SD	AVG	SD	AVG	AVG
64	0.17	0.04	0.08	0.03	0.02	0.00	7.66	3.65
128	2.66	0.04	0.96	0.04	0.18	0.02	15.06	5.46
256	50.62	0.85	22.79	1.77	2.89	0.22	17.53	7.89
512	1036.81	3.56	554.05	23.42	67.78	3.87	15.30	8.17
1024	22812.61	184.43	14938.05	107.67	1697.77	158.55	13.44	8.80
2048	—	—	—	—	44025.15	1493.82	—	—

Table 9. Extended Tests of Solution Phase Runtimes (s) (Statistics Apply to the Sum of 50 RHS Vectors Solved per Repetition)

n	Doolittle-LU		Q-Mat-Adj		REF-LU		Dool/REF	Q-Mat/REF
	AVG	SD	AVG	SD	AVG	SD	AVG	AVG
64	1.135	0.016	0.014	0.003	0.033	0.009	34.06	0.42
128	11.000	0.479	0.062	0.002	0.206	0.031	53.40	0.30
256	103.281	0.358	0.352	0.022	2.119	0.278	48.74	0.17
512	1018.712	3.154	2.184	0.022	26.828	2.005	37.97	0.08
1024	10896.450	66.268	15.455	0.401	340.876	10.104	31.97	0.05
2048	—	—	—	—	4307.606	310.718	—	—

on 15 separate machines with the specifications described in Section 3.1 (but with different levels of wear and tear); by contrast, each of the previous experiments performed its full set of repetitions on single machines. It is important to reemphasize that the solution phase statistics apply to the sum of 50 distinct RHS vectors per each of 30 repetitions (to obtain meaningful timing results for small n). Hence, the reported values should be divided by 50 to obtain a better approximation of the runtimes required to solve a single RHS vector. The solution phase runtimes values are rounded to three decimal places to account for the lower magnitudes associated with smaller basis dimensions.

The results hint at the asymptotic relationships between the runtimes of the three approaches. To bring these relationships to light, Figures 2 and 3 provide log-log plots of the average runtimes reported in Tables 8 and 9, respectively, in seconds (s). To be precise, the plotted values are the base-2 log average runtimes (shifted by .001s) corresponding to the base-2 log of each value of n . From Figure 2, it is possible to discern the point at which slower-memory operations begin to have a substantial effect on each construction phase algorithm. The slowdown occurs sooner for Doolittle-LU and then Q-Mat-Adj since they require approximately two and three times the memory of REF-LU (see Sections 4.2.2 and 5.2.2). Figure 2 also illustrates the gradually increasing ratio of the REF-LU runtimes relative to those of Q-Mat-Adj. A similar but reversed relationship between the solution phase runtimes of both approaches can be observed in Figure 3, where Doolittle-LU again achieves the worst performance. Most interestingly, it appears that Doolittle-LU would prevail over Q-Mat-Adj for large n in terms of construction phase runtimes, as intimated by Figure 2. The likely reason for this trend is that the costs of Q-Mat-Adj's growing memory requirements will outweigh the costs of Doolittle-LU's rational arithmetic operations in the long run. This expectation is echoed by the nearly identical slopes of their log-log plots (excluding small n values). Expressly, from $n = 2^4$ to $n = 2^{10}$, the Figure 2 log-log plots can be interpolated via lines $y_{\text{Doolittle-LU}} = 4.02(\log_2 n - 3) - 14.18$, $y_{\text{Q-Mat-Adj}} = 3.96(\log_2 n - 3) - 14.805$, and

Fig. 2. Log-log plot for the extended tests of construction phase runtimes.

Fig. 3. Log-log plot for the extended tests of solution phase runtimes.

$y_{\text{REF-LU}} = 3.48(\log_2 n - 3) - 14.99$. For completeness, from $n = 2^3$ to $n = 2^{10}$, the Figure 3 log-log plots can be interpolated via lines $y_{\text{Doolittle-LU}} = 3.11(\log_2 n - 3) - 11.90$, $y_{\text{Q-Mat-Adj}} = 2.63(\log_2 n - 3) - 14.12$, and $y_{\text{REF-LU}} = 2.01(\log_2 n - 3) - 13.20$.

6 CONCLUSION

This article demonstrates that when the underlying SLEs are dense, the REF factorization framework outperforms the exact rational arithmetic LU factorization approach, the latter of which is a tool utilized by both state-of-the-art exact mathematical programming solvers QSOpt_ex (Applegate et al. 2007b) and SoPlex (Gleixner et al. 2015; Wunderling 1996) for basic solution validation. Indeed, in the featured experiments the factors of improvement were as high as 22 and 55 for the construction (i.e., factorization) phases and solution phases, respectively. Moreover, the REF LU factorization used practically half the memory required by the exact rational arithmetic

Doolittle and Crout factorizations. This work also develops and tests a streamlined version of the Q-matrix approach tailored to basic solution validation. Related experiments reveal that the REF LU factorization can be constructed close to nine times faster than the Q-matrix (i.e., the adjunct matrix of the basis) for the larger tested instances. However, these experiments also show that the Q-matrix solution phase can be more than 22 times faster than REF forward and backward substitution over the same instances. Nevertheless, based on the considerations discussed in the next paragraph, the REF factorization framework is preferable to these two alternative direct solution approaches for basic solution validation when dealing with dense instances.

The REF factorization is preferable to the streamlined Q-matrix approach owing to four key considerations. First, construction phase runtimes are significantly greater in magnitude than solution phase runtimes for each method; in the featured experiments this magnitude difference widens as n increases, which is explained by the fact that they entail $O(n^3)$ and $O(n^2)$ operations, respectively. Second, validating a basic solution requires solving for only two RHS vectors at a time (corresponding to the primal and dual solution validations), meaning the computational advantage of the Q-matrix approach has limited impact overall. Third, the experiment results suggest that, based on its growing memory requirements for larger matrix dimensions, the adjunct matrix would eventually take longer to construct than the exact rational arithmetic LU factorizations. Fourth, we project that the computational advantages of the REF factorization framework would increase when dealing with sparse instances. The reason is that LU factorization algorithms can be designed to preserve and exploit sparsity, in contrast to the construction and deployment of the adjunct matrix, which is a scaled version of the generally dense inverse matrix.

It is important to acknowledge that, since this article focuses on solving dense SLEs but most LPs and MIPs are sparse, further steps must be undertaken before the potential of the REF framework can be realized in state-of-the-art exact LP and MIP solvers. These steps include developing a sparse REF factorization framework and performing corresponding tests on realistic sparse LP instances. These contributions are significant in their own right, and we plan to pursue them in our future work. As a final remark, we predict that the computational performance of every method herein tested would be proportionally enhanced by the use of the EGLib library (Espinoza and Goycoolea 2003) for fast memory allocation and by other technical advancements to our codes. Hence, it is highly likely that the reported performance of each tested unlimited-precision method could be improved fairly evenly in more sophisticated implementations. For this reason, the computational performance realized by the REF factorization framework can be regarded as a starting point for more finely tuned implementations.

Interested researchers are encouraged to use or modify our general purpose REF-LU codes (available via GitHub at <https://github.com/adolfoescobedo/REF-LU>) and to use them in their work by citing Escobedo and Moreno-Centeno (2015) and this article.

ACKNOWLEDGMENTS

The authors would like to take this opportunity to thank the editors and anonymous reviewers for their constructive feedback and helpful suggestions, which helped enhance the quality of this article.

REFERENCES

- David L. Applegate, William Cook, Sanjeeb Dash, and Daniel G. Espinoza. 2007a. Exact solutions to linear programming problems. *Op. Res. Lett.* 35, 6 (2007), 693–699.
- David L. Applegate, Sanjeeb Dash, William Cook, and Daniel G. Espinoza. 2007. QSOpt_ex Rational LP Solver. Retrieved July 1, 2016 from <http://www.dii.uchile.cl/~daespino>.
- David-Olivier Azulay and Jean-Francois Pique. 2001. A revised simplex method with integer Q-matrices. *ACM Trans. Math. Softw.* 27, 3 (2001), 350–360.

- David H. Bailey and Jonathan M. Borwein. 2015. High-precision arithmetic in mathematical physics. *Mathematics* 3, 2 (2015), 337–367.
- Erwin H. Bareiss. 1968. Sylvester’s identity and multistep integer-preserving Gaussian elimination. *Math. Comp.* 22, 103 (1968), 565–578.
- Erwin H. Bareiss. 1972. Computational solutions of matrix problems over an integral domain. *IMA J. Appl. Math.* 10, 1 (1972), 68–104.
- Gleb Beliakov and Yuri Matiyasevich. 2016. A parallel algorithm for calculation of determinants and minors using arbitrary precision arithmetic. *BIT Numer. Math.* 56, 1 (2016), 33–50.
- Zhuliang Chen and Arne Storjohann. 2005. A BLAS based C library for exact linear algebra on integer matrices. In *Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation*. ACM, 92–99.
- William Cook, Thorsten Koch, Daniel Steffy, and Kati Wolter. 2011. An exact rational mixed-integer programming solver. *Integer Programming and Combinatorial Optimization*. Springer, Berlin, Heidelberg, 104–116.
- William Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. 2013. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Math. Program. Comput.* 5, 3 (2013), 305–344.
- William Cook and Daniel E. Steffy. 2011. Solving very sparse rational systems of equations. *ACM Trans. Math. Softw.* 37, 4 (2011), 39.
- Payman Dehghanian, Yaping Wang, Gurunath Gurrula, Erick Moreno-Centeno, and Mladen Kezunovic. 2015. Flexible implementation of power system corrective topology control. *Electr. Power Syst. Res.* 128 (2015), 79–89.
- Marcel Dhiflaoui, Stefan Funke, Carsten Kwappik, Kurt Mehlhorn, Michael Seel, Elmar Schömer, Ralph Schulte, and Dennis Weber. 2003. Certifying and repairing solutions to large LPs how good are LP-solvers? In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 255–256.
- John D. Dixon. 1982. Exact solution of linear equations using P -adic expansions. *Numer. Math.* 40, 1 (1982), 137–141.
- Jack Edmonds. 1967. Systems of distinct representatives and linear algebra. *J. Res. Natl. Bur. Stand., Sect. B* 71 (1967), 241–245.
- Jack Edmonds and J.-F. Maurras. 1997. Note sur les Q -matrices d’Edmonds. *RAIRO. Rech. Op.* 31, 2 (1997), 203–209.
- Adolfo R. Escobedo. 2016. *Foundational Factorization Algorithms for the Efficient Roundoff-Error-Free Solution of Optimization Problems*. Ph.D. Dissertation. Department of Industrial and Systems Engineering, Texas A&M University, College Station, TX.
- Adolfo R. Escobedo and Erick Moreno-Centeno. 2015. Roundoff-error-free algorithms for solving linear systems via Cholesky and LU factorizations. *INFORMS J. Comput.* 27, 4 (2015), 677–689.
- Adolfo R. Escobedo and Erick Moreno-Centeno. 2017. Roundoff-error-free basis updates of LU factorizations for the efficient validation of optimality certificates. *SIAM J. Matrix Anal. Appl.* 38, 3 (2017), 829–853.
- Adolfo R. Escobedo, Erick Moreno-Centeno, and Kory W. Hedman. 2014. Topology control for load shed recovery. *IEEE Trans. Power Syst.* 29, 2 (2014), 908–916.
- Daniel G. Espinoza and Marcos Goycoolea. 2003. EGlib, Efficient General Library. Retrieved July 1, 2016 from http://dii.uchile.cl/~daespino/EGlib_doc.
- George E. Forsythe and Cleve B. Moler. 1967. *Computer Solution of Linear Algebraic Systems*, Vol. 7. Prentice-Hall, Englewood Cliffs, NJ.
- Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. 2007. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.* 33, 2 (2007), 13.
- Gerald Gamrath, Tobias Fischer, Tristan Gally, Ambros M. Gleixner, Gregor Hendel, Thorsten Koch, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, et al. 2016. The SCIP optimization suite. *ZIB Rep.* (2016), 15–60.
- Bernd Gärtner. 1999. Exact arithmetic at low cost—a case study in linear programming. *Comput. Geom.* 13, 2 (1999), 121–139.
- John R. Gilbert. 1994. Predicting structure in sparse matrix computations. *SIAM Journal on Matrix Analysis and Applications* 15 (1994), 62–79.
- Ambros Gleixner, Matthias Miltenberger, and Benjamin Müller. 2015. SoPlex: the sequential object-oriented simplex class library, version 2.2. Retrieved July 1, 2016 from <http://soplex.zib.de>.
- Ambros M. Gleixner. 2015. *Exact and Fast Algorithms for Mixed-Integer Nonlinear Programming*. Ph.D. Dissertation. Technische Universität Berlin.
- Ambros M. Gleixner, Daniel E. Steffy, and Kati Wolter. 2012. Improving the accuracy of linear programming solvers with iterative refinement. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*. ACM, 187–194.
- Ambros M. Gleixner, Daniel E. Steffy, and Kati Wolter. 2016. Iterative refinement for linear programming. *INFORMS J. Comput.* 28, 3 (2016), 449–464.
- Torbjörn Granlund et al. 2015. GNU MP 6.0 Multiple Precision Arithmetic Library, Samurai Media Limited.
- Tudor Jebelean. 1997. Practical integer division with Karatsuba complexity. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*. ACM, 339–341.

- Erich Kaltofen and B. David Saunders. 1991. On Wiedemann's method of solving sparse linear systems. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*. Springer, 29–38.
- Ed Klotz. 2014. Identification, assessment, and correction of Ill-conditioning and numerical instability in linear and integer programs. *TutORials in Operations Research: Bridging Data and Decisions*. Informs, 54–108.
- Thorsten Koch. 2004. The final NETLIB-LP results. *Op. Res. Lett.* 32, 2 (2004), 138–142.
- Na Li, Yousef Saad, and Edmond Chow. 2003. Crout versions of ILU for general sparse matrices. *SIAM J. Sci. Comput.* 25, 2 (2003), 716–728.
- Ding Ma and Michael A. Saunders. 2015. Solving multiscale linear programs using the simplex method in quadruple precision. In *Numerical Analysis and Optimization*. Springer, 223–235.
- Hans D. Mittelmann. 2006. LPtestset. Retrieved July 1, 2016 from <http://plato.asu.edu/ftp/lptestset>.
- Bruce A. Murtagh. 1981. *Advanced Linear Programming: Computation and Practice*. McGraw-Hill International Book Co.
- Arnold Neumaier and Oleg Shcherbina. 2004. Safe bounds in linear and mixed-integer linear programming. *Math. Program.* 99, 2 (2004), 283–296.
- Scott A. Sarra. 2011. Radial basis function approximation methods with extended precision floating point arithmetic. *Eng. Anal. Bound. Elem.* 35, 1 (2011), 68–76.
- Alexey Solovyev and Thomas C. Hales. 2011. Efficient formal verification of bounds of linear programs. In *International Conference on Intelligent Computer Mathematics*. Springer, 123–132.
- Daniel E. Steffy. 2011a. Exact solutions to linear systems of equations using output sensitive lifting. *ACM Commun. Comput. Algebra* 44, 3/4 (2011), 160–182.
- Daniel E. Steffy. 2011b. *Topics in Exact Precision Mathematical Programming*. Ph.D. Dissertation. Georgia Institute of Technology.
- Zhendong Wan. 2006. An algorithm to solve integer linear systems exactly using numerical methods. *J. Symbol. Comput.* 41, 6 (2006), 621–632.
- Douglas Wiedemann. 1986. Solving sparse linear equations over finite fields. *IEEE Trans. Inform. Theor.* 32, 1 (1986), 54–62.
- James Hardy Wilkinson. 1963. *Rounding Errors in Algebraic Processes*. Prentice-Hall, Englewood Cliffs, NJ.
- Roland Wunderling. 1996. *Paralleler und Objektorientierter Simplex-Algorithmus*. Ph.D. Dissertation. Technische Universität Berlin.

Received August 2016; revised March 2018; accepted March 2018