Please cite as:

Christopher Lourenco, Jinhao Chen, Erick Moreno-Centeno, and Timothy A. Davis. 2022. Algorithm 1021: SPEX Left LU, Exactly Solving Sparse Linear Systems via a Sparse Left-looking Integer-preserving LU Factorization. *ACM Trans. Math. Softw.* 48, 2, Article 20 (May 2022), 23 pages.
https://doi.org/10.1145/3519024

Algorithm 1XXX: SPEX Left LU, Exactly Solving Sparse Linear Systems via a Sparse Left-Looking Integer-Preserving LU Factorization*

CHRISTOPHER LOURENCO, United States Naval Academy, USA

JINHAO CHEN, Texas A&M University, USA

ERICK MORENO-CENTENO, Texas A&M University, USA and Instituto Tecnológico Autónomo de México, México

TIMOTHY A. DAVIS, Texas A&M University, USA

SPEX Left LU is a software package for exactly solving unsymmetric sparse linear systems. As a component of the sparse exact (SPEX) software package, SPEX Left LU can be applied to any input matrix, A, whose entries are integral, rational, or decimal, and provides a solution to the system $Ax = b$ which is either exact or accurate to user-specified precision. SPEX Left LU preorders the matrix A with a user-specified fill-reducing ordering and computes a left-looking LU factorization with the special property that each operation used to compute the L and U matrices is integral. Notable additional applications of this package include benchmarking the stability and accuracy of state-of-the-art linear solvers, and determining whether singular-to-double-precision matrices are indeed singular. Computationally, this paper evaluates the impact of several novel pivoting schemes in exact arithmetic, benchmarks the exact iterative solvers within Linbox, and benchmarks the accuracy of MATLAB sparse backslash. Most importantly, it is shown that SPEX Left LU outperforms the exact iterative solvers in run time on *easy* instances and in stability as the iterative solver fails on a sizeable subset of the tested (both easy and hard) instances. The SPEX Left LU package is written in ANSI C, comes with a MATLAB interface, and is distributed via GitHub, as a component of the SPEX software package, and as a component of SuiteSparse.

CCS Concepts: • **Mathematics of computing → Solvers**; **Computations on matrices**; **Mathematical software performance**;

Additional Key Words and Phrases: exactly solving linear systems, sparse linear systems, sparse matrix algorithms, roundoff errors, exact matrix factorization

ACM Reference Format:
Christopher Lourenco, Jinhao Chen, Erick Moreno-Centeno, and Timothy A. Davis. 2022. Algorithm 1XXX: SPEX Left LU, Exactly Solving Sparse Linear Systems via a Sparse Left-Looking Integer-Preserving LU Factorization. *ACM Trans. Math. Softw.* 1, 1, Article 1 (July 2022), 28 pages.
https://doi.org/0000001.0000001

*This paper is dedicated to the memory of René Mario Montante-Pardo

Authors' addresses: Department of Mathematics, US Naval Academy, Annapolis MD, email: lourenco@usna.edu; Department of Industrial and Systems Engineering, Texas A&M University, College Station TX, email: emc@tamu.edu; Department of Computer Science and Engineering, Texas A&M University, College Station TX, email: jinhchen@tamu.edu, davis@tamu.edu.

Manuscript submitted to ACM

1 OVERVIEW

Exactly solving sparse systems of linear equations (SLEs) is a key subroutine of algorithms used to solve problems arising in various fields including number theory [Dixon 1982; Wiedemann 1986], mathematical proofs [Hales 2005], computational geometry [Burton and Ozlen 2012; Gärtner 1999], and exact linear/integer programming [Gleixner 2015; Steffy 2011]. Moreover, solving a sparse SLE may require extended precision due to numerical instability [Higham 2002; Klotz 2014] or a poorly scaled or highly ill-conditioned input matrix [Golub and Van Loan 2012; Horn and Johnson 2012]. Lourenco et al. [2019] derived the sparse left-looking integer-preserving (SLIP) LU factorization, which exactly solves the sparse SLE, $A\mathbf{x} = \mathbf{b}$ exclusively using integer-arithmetic; thereby ensuring that the final solution to the system is roundoff-error-free.

1.1 Contributions

We develop the Sparse Exact (SPEX) Left LU software package which implements the SLIP LU factorization in ANSI C along with a MATLAB interface. The package performs all internal operations on full-precision integers (via the GNU Multiple Precision Arithmetic (GMP) Library [Granlund et al. 2015]). SPEX Left LU either gives the exact solution to the input linear system exactly (as a rational vector via the GMP library) or to any user defined precision (either double or variable precision floating point via the GNU Multiple Precision Floating-Point Reliable (MPFR) library [Fousse et al. 2007]). In the case of (double precision) floating point input, the input is scaled to integers by multiplying by 10^{16} (which is chosen, as explained briefly below, due to the fact that machine precision is $2.2204 * 10^{-16}$); then, the solution is exact to the scaled system. This makes sense as, given two numbers, whose magnitudes differ by $\approx 10^{16}$ or more, the smaller number is, effectively, zero (i.e., $x = 10^{18}$, $y = 1$, $x + y == x$; and likewise, $x = 1$, $y = 10^{-18}$, $x + y == x$.) In order to obtain higher precision (i.e., to preserve the precision of all the matrix's entries regardless of how relatively tiny they are compared to the other matrix's entries), the user can use MPFR input which allows (floating-point) numbers of arbitrary digits of precision with "exact" conversion (thus, the exact number of digits are preserved).

The software package presented in this paper provides the first, commercial quality implementation of a direct method to exactly solve sparse linear systems solely using integer arithmetic. The software is extremely reliable in two aspects: (1) the code itself has undergone 100% test coverage along with scaffolding code to test loop invariants and data sanity and (2) in terms of algorithmic reliability, as it guarantees to provide, with probability **1**, an exact solution unlike competitor approaches as further discussed in the computational results. To improve usability, the API of the code was completely redesigned, the vast majority of the functions were rewritten since the publication of [Lourenco et al. 2019], and an easy to use MATLAB interface is provided so that users can solve the linear system $A\mathbf{x} = \mathbf{b}$ and obtain output in either double precision, variable precision, or a string of numerators and denominators through a single line of MATLAB code, `x = SPEX_Left_LU_backslash(A,b)`. Additionally, all user visible functions within the code have been extensively documented. As a component of the SPEX software package, we envision SPEX Left LU to be the first in a line of forthcoming codes to solve any sparse linear system exactly.

Moreover, an extra improvement contained in our package has ramifications beyond exactly solving sparse linear systems. As mentioned above, SPEX Left LU is based on the GNU GMP [Granlund et al. 2015] and GNU MPFR [Fousse et al. 2007] libraries, which are two of the most robust and widely used libraries for arbitrary bit-length integer, rational, and extended floating point operations. However, despite the widespread use of these libraries, they handle memory errors in an extremely low-level manner: for instance, when either library runs out of memory when performing arithmetic operations (such as $a = b \times c$), they will either segmentation fault or abort. While this approach is valid for

some software, other packages such as MATLAB and CPLEX require more robust error handling; thus, due to this issue, GMP and MPFR are unsuitable to be used (or included within) this type of software packages. To alleviate this issue, we developed generic wrappers for all GNU GMP/MPFR functions so that errors such as out of memory do not lead to segmentation fault or abort. When coupled with our wrappers, GMP and MPFR (and thus any software that uses our interface to GMP/MPFR) will simply indicate the memory issue and close cleanly. Note that, if a developer desires to use our GMP/MPFR wrappers in other applications, they can be used independently of the SPEX factorization framework.

The paper concludes with a robust trio of computational studies. First, we show that our exact factorization performs best when using a counter-intuitive partial pivoting scheme in which the k^{th} pivot element is selected as the entry of *lowest magnitude* in the k^{th} column. This is in contrast to floating-point left-looking algorithms such as Gilbert and Peierls [Gilbert and Peierls 1988] or KLU [Davis and Palamadai Natarajan 2010] which must perform partial pivoting on entries with *large* magnitude in order to maintain floating-point numerical stability. Second, we utilize SPEX Left LU to benchmark the accuracy of the state-of-the-art sparse matrix solver within MATLAB, illustrating that this commercial solver may produce inaccurate solutions for about 3% of real world "well behaved" instances, and up to 37% of real world, singular-to-double-precision instances. Third, we compare SPEX Left LU to an alternate method of exactly solving sparse linear systems, iterative black-box approaches. These tests show that our factorization outperforms in run time the iterative methods for *easy* instances while the iterative methods outperform SPEX Left LU on the *hard* instances. As a result, this paper presents evidence that SPEX Left LU is among the state-of-the-art methods to exactly, efficiently, and reliably solve sparse linear systems; specifically, on small and medium sized instances without an abundance of fill.

1.2 Organization

The remainder of this paper is organized as follows. Section 2 gives a brief overview of the underlying algorithms used in the SPEX Left LU package. Section 3 describes the implementation of the SPEX Left LU package. Section 4 computationally compares pivoting schemes for the exact factorization, benchmarks the accuracy of MATLAB sparse backslash, and compares SPEX Left LU to alternate iterative exact methods for solving sparse linear systems. Finally, Section 5 concludes the work.

2 THE SLIP LU FACTORIZATION

This section provides a brief overview of the SLIP LU factorization, the basis of SPEX Left LU. For simplicity, the description below assumes that no row and column permutations are applied to the matrix A, an assumption that is removed for the remainder of the paper. See [Lourenco et al. 2019] for an in-depth discussion and theoretical derivation of the factorization.

Based on integer-preserving Gaussian elimination (IPGE) [Bareiss 1968; Edmonds 1967; Montante-Pardo and Méndez-Cavazos 1977], its improvements [Lee and Saunders 1995], and left-looking LU factorization [Gilbert and Peierls 1988], the SLIP LU factorization expands the (dense) roundoff-error-free (REF) LU factorization [Escobedo and Moreno-Centeno 2015] to the sparse case. Given an input matrix, $A \in \mathbb{Z}^{n\times n}$, and right hand side vector, $\mathbf{b} \in \mathbb{Z}^{n\times 1}$, SLIP LU computes the sparse factorization $A = LDU$, where $L, U \in \mathbb{Z}^{n\times n}$ and D is a diagonal matrix. Note that, if A is rational or decimal, the SLIP LU package makes A integral by multiplying its entries by the least common multiple or appropriate power of 10, respectively. Then, using specialized sparse integer-preserving forward and backward substitution algorithms, the SLIP LU factorization solves the system $LDU\mathbf{x} = \mathbf{b}$, where $\mathbf{x} \in \mathbb{Q}^{n\times 1}$, using only integer-arithmetic. The key property of SLIP LU is that, in the bit-complexity model, it solves the linear system $A\mathbf{x} = \mathbf{b}$ in time proportional to the arithmetic work. This means that all of the overhead related to data structures and ancillary operations is

asymptotically dominated by the cost of the arithmetic operations required to compute the factorization. Notably, this property is not trivial in sparse-matrix algorithms; indeed, to the best of our knowledge, there are only two other sparse factorizations that achieve this asymptotically efficient complexity bound (when implemented with any fixed precision type): Gilbert and Peierls' left-looking LU factorization [Gilbert and Peierls 1988] and Liu's up-looking Cholesky factorization [Liu 1991]. In contrast, exact factorization approaches based on arbitrary precision rational arithmetic (opposed to our factorization which is based on arbitrary precision integer arithmetic) for solving sparse linear systems cannot achieve such asymptotic efficiency due to their usage of ancillary, computationally expensive greatest common divisor operations.

Prior to reviewing the factorization, we introduce the following notation:

Definition 2.1 (IPGE Algorithm). Let $A^{(k)}$ denote the k^{th} iteration IPGE matrix, for $0 \leq k \leq n$, with $A^{(0)} = A$. Additionally, let $a_{i,j}^{(k)}$ denote the individual entries of $A^{(k)}$ for $1 \leq i \leq n$, $1 \leq j \leq n$, and $0 \leq k \leq n$. At iteration k, the IPGE algorithm computes entry $a_{i,j}^{(k)}$ as follows:

$$a_{i,j}^{(k)} = \begin{cases} a_{i,j}^{(k-1)} & \text{if } i = k, \\ \dfrac{a_{k,k}^{(k-1)} a_{i,j}^{(k-1)} - a_{k,j}^{(k-1)} a_{i,k}^{(k-1)}}{a_{k-1,k-1}^{(k-2)}} & \text{otherwise.} \end{cases} \tag{1}$$

Definition 2.2. Let $\rho^{(k)}$ denote the k^{th} pivot element chosen during the SLIP LU factorization, for $0 \leq k \leq n$, with $\rho^{(0)} = 1$. In terms of IPGE entries $\rho^{(k)} = a_{k,k}^{(k-1)}$.

Definition 2.3. Let $L^{(k)}$ and $D^{(k)}$ be the k^{th} left-looking L and D matrices, respectively, for $k = 0, \ldots, n$. Specifically, $L^{(k)}$ and $D^{(k)}$ are the first k completed columns of L and D augmented by the last $n - k$ columns of I and $(1/\rho^{(k)})I$, respectively.

Definition 2.4. Let $A(:, k)$ denote the k^{th} column of the matrix A, for $1 \leq k \leq n$.

The SLIP LU factorization is a left-looking LU factorization algorithm; thus, at iteration k, it computes the k^{th} column of the matrices L and U. To do so, it solves partial lower triangular linear systems of the form $L^{(k-1)}D^{(k-1)}\mathbf{x} = A(:, k)$ for $k = 1, \ldots, n$. Formally, the solution to Equation (2) yields the k^{th} column of L and U.

$$L^{(k-1)}D^{(k-1)}\mathbf{x} = \left[\begin{array}{cccc|c} \frac{1}{\rho^{(0)}} & 0 & 0 & 0 & \\ \frac{l_{2,1}}{\rho^{(1)}} & \frac{1}{\rho^{(1)}} & 0 & 0 & \\ \vdots & & \ddots & & \mathbf{0} \\ \frac{l_{k-1,1}}{\rho^{(1)}} & \cdots & \cdots & \frac{1}{\rho^{(k-2)}} & \\ \hline \frac{l_{k,1}}{\rho^{(1)}} & \cdots & \cdots & \frac{l_{k,k-1}}{\rho^{(k-2)}\rho^{(k-1)}} & \\ \vdots & & & \vdots & \frac{1}{\rho^{(k-1)}}\mathbf{I} \\ \frac{l_{n,1}}{\rho^{(1)}} & \cdots & \cdots & \frac{l_{n,k-1}}{\rho^{(k-2)}\rho^{(k-1)}} & \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,k}^{(0)} \\ a_{2,k}^{(0)} \\ \vdots \\ a_{n,k}^{(0)} \end{bmatrix}. \tag{2}$$

In order to solve Equation (2), a sparse REF lower triangular solve algorithm, Algorithm 1, was derived. Algorithm 1 solves this system in two phases: symbolic and numeric.

Symbolically, Algorithm 1 first computes the nonzero pattern of $\mathbf{x}$, denoted $\mathcal{X}$, via a graph search algorithm. Specifically, this algorithm operates on a graph of n nodes where a directed edge (i, j) exists if $l_{j,i}$ is nonzero. It performs

a sequence of depth first searches, each rooted on each of the nonzeros in $A(:,k)$. Consequently, this algorithm is referred to as obtaining the *reach* of the k^{th} column of A on the graph of L and is denoted $\mathcal{X} = Reach_{G_L}(A(:,k))$. This algorithm outputs the nonzeros in topological order, which is sufficient for traditional lower triangular solve algorithms [Gilbert and Peierls 1988]. However, in order to preserve integrality, SLIP LU requires that this nonzero pattern is sorted with respect to the row indices.

Interestingly, unlike floating-point left-looking LU, the cost of the integer-arithmetic operations dominate the sort; thus, as per Theorem 4.8 of [Lourenco et al. 2019], this sort does not impact the asymptotic efficiency of SLIP LU.

With the nonzero pattern $\mathcal{X}$ in hand, Algorithm 1 next computes the numeric values of $\mathbf{x}$. This is done via a sequence of two operations referred to as a History update and IPGE update. A History update consists of multiplying x_j by the the current pivot and dividing it by a previous pivot; an operation which is essential to exploiting sparsity in IPGE. An IPGE update consists of a single step of IPGE (i.e., Equation 1) applied to entry x_j. This algorithm is the basis of the numeric factorization of SLIP LU; thus, it is presented below for completeness.

Algorithm 1 Sparse REF Lower Triangular Solve

1: $x = A(:,k)$
2: $\mathcal{X} = Reach_{G_L}(A(:,k))$
3: **sort**($\mathcal{X}$)
4: Initialize history vector: $h_j = 0\ \forall j \in \mathcal{X}$
5: **for** $j \in \mathcal{X}$ **do**
6: **if** $j < k$ **then**
7: **if** $h_j < j-1$ **then**
8: History update: $x_j = \frac{x_j \rho^{(j-1)}}{\rho^{(h_j)}}$
9: **end if**
10: **for** $i > j$ and $l_{i,j} \neq 0$ **do**
11: **if** $h_i < j-1$ **then**
12: History update: $x_i = \frac{x_i \rho^{(j-1)}}{\rho^{(h_i)}}$
13: **end if**
14: IPGE update: $x_i = \frac{\rho^{(j)} x_i - l_{i,j} x_j}{\rho^{(j-1)}}$
15: History vector update: $h_i = j$
16: **end for**
17: **else**
18: **if** $h_j < k-1$ **then**
19: History update: $x_j = \frac{x_j \rho^{(k-1)}}{\rho^{(h_j)}}$
20: **end if**
21: **end if**
22: **end for**

3 FEATURES OF THE SPEX LEFT LU PACKAGE

This section gives a brief overview of the features of SPEX Left LU. For an in-depth discussion of the C and MATLAB interfaces to this package, we refer the reader to the user guide included with the software package.

3.1 Software Dependencies

Both the C and MATLAB implementation of SPEX Left LU require the installation of four external software libraries. The first two libraries, the GNU multiple precision arithmetic library (GMP) [Granlund et al. 2015] and GNU multiple precision floating-point reliable library (MPFR) [Fousse et al. 2007], are distributed independently from SPEX Left LU. The other two libraries, the approximate minimum degree (AMD) ordering [Amestoy et al. 2004], and column approximate minimum degree (COLAMD) ordering [Davis et al. 2004], are distributed along with SPEX Left LU; however, they may also be independently obtained via SuiteSparse [Davis et al. 2014].

3.2 Functionality of SPEX Left LU

Using the SPEX Left LU package to exactly solve a sparse linear system $A\mathbf{x} = \mathbf{b}$ comprises five phases: input, column preordering, factorization, forward/backward substitution, and output.

The input phase of SPEX Left LU creates a scaled (if necessary) copy of the user's input matrix to use for all further routines. This copy is stored in sparse compressed column form and its numeric entries are full precision integers via GMP's big integer (`mpz_t`) data structure. The user's input matrix may be stored in either triplet, sparse compressed column, or dense format. The numeric entries in the user's matrix may be read from: GMP big integer (`mpz_t`), GMP rational numbers (`mpq_t`), MPFR variable precision floating-point numbers (`mpfr_t`), double precision numbers, or 64 bit integers (`int64_t`). For all of the inexact input data types, the values in the matrix are assumed to be accurate to the defined precision. All matrices in this package are stored in a `SPEX_matrix` struct.

The column preordering phase of SPEX Left LU reorders the columns of the matrix A in order to reduce the number of nonzeros in the ensuing L and U factors. By default SPEX Left LU uses the COLAMD ordering [Davis et al. 2004]; however, AMD [Amestoy et al. 2004] can be used, or the matrix can be factorized with no column reordering, instead.

The third phase of SPEX Left LU is the integer-preserving factorization. It computes the factorization $LDU = PAQ$ by performing n iterations of the sparse REF lower triangular solve, Algorithm 1, described in Section 2. By default, the pivot matrix P is selected via a partial pivoting scheme in which the k^{th} pivot is the diagonal entry if its magnitude is smallest in column k, otherwise it is the smallest entry in column k. SPEX Left LU allows various other pivoting schemes; refer to the user guide for details.

The forward/backward substitution phase of SPEX Left LU uses the integral factorization to solve the linear system $A\mathbf{x} = \mathbf{b}$. This is done via REF forward substitution [Escobedo and Moreno-Centeno 2015; Lourenco et al. 2019], which entails solving the system $LD\mathbf{y} = P\mathbf{b}$, and backward substitution which entails solving the system $U\mathbf{z} = det(A)\mathbf{y}$, where $det(A)$ is the determinant of the A matrix (which is the n^{th} pivot of the factorization). After completing backward substitution, the exact rational solution of the system $A\mathbf{x} = \mathbf{b}$ is given as $\mathbf{x} = \mathbf{z}/det(A)$. At this point, $\mathbf{x}$ is rational and guaranteed to be exact.

The output phase of SPEX Left LU returns the solution vector(s) $\mathbf{x}$ to the user. In C, the user can obtain $\mathbf{x}$ in rational form (via GMP `mpq_t` data type) or it can be rounded to either double precision or a user specified floating point precision (via the MPFR `mpfr_t` data type). In MATLAB, the user can obtain $\mathbf{x}$ in either double precision, extended precision via MATLAB `vpa`, or a cell array of rational strings. Note that roundoff error in floating-point types is only accrued in this final conversion from rational arithmetic. SPEX Left LU utilizes higher precision for this conversion; thus, double precision output is accurate to machine roundoff (approximately $2^{-52} \approx 2.22 * 10^{-16}$ [Kahan 1996]), while multiple precision output is accurate to the user's specified precision.

3.3 Improvements to GMP and MPFR made in SPEX

The GMP and MPFR libraries may suffer from run-time errors due to lack of memory or invalid user input. By default, both of these libraries abort the user's application if any internal routine fails, which is not acceptable in a robust end-user application. Thus, to improve the stability of these two libraries, we developed a set of wrappers for all utilized GMP and MPFR functions which properly handle out of memory conditions or errors due to user input via the ANSI C longjmp function and a global memory heap manager. As a result, the SPEX package will not crash due to memory issues or invalid user input. Our wrapper class for all utilized GMP and MPFR functions is described in `SPEX/SPEX_Util/Source/SPEX_gmp.c` and `Include/SPEX_Util.h`. Our implementation can be extended to any other GMP or MPFR function, by following the template given in those two files. Moreover, these wrapper functions can be used independently of the SPEX factorization routines; for this, a developer only needs to include `SPEX_Util.h` in their code to use out GMP/MPFR wrappers in their applications.

4 COMPUTATIONAL TESTS

This section presents a computational study of the SPEX Left LU package. Section 4.1 describes the computing environment used. Section 4.2 discusses the set of instances used for testing. Section 4.3 computationally compares various pivoting schemes within SPEX Left LU. Section 4.4 benchmarks the accuracy of MATLAB sparse backslash. Finally, Section 4.5 compares the SPEX Left LU factorization to the (exact) iterative methods within Linbox [Dumas et al. 2002].

4.1 Computing Environment

The experiments conducted in Sections 4.3 and 4.5.2 measure run time and were coded in C and performed on a computing node running CentOS 7 with 192GB of RAM shared by two 2.0 GHz Intel Xeon 6138 processors with 20 cores. The experiments conducted in Sections 4.4 and 4.5.1 measure accuracy and were performed on both the aforementioned computing node as well as in MATLAB R2020a on a computer running Ubuntu 18.04 with 32GB of RAM using a 3.7 GHz Intel Core i9-10900K CPU with 20 cores.

Throughout these computational tests, we use Dolan and MorÃĺ [Dolan and Moré 2002] performance profiles when comparing competing algorithms/approaches. Briefly, a performance profile is a tool which takes into account both the number of instances solved as well as the cost required to solve each instance. The performance of each algorithm corresponds to a curve on a graph, where each point on the curve is what percentage of instances (y-axis) the algorithm solved within a time-multiple (x-axis) of the fastest solution time (among all algorithms) for each instance. An important property of performance profiles is that they are insensitive to the relative difficulty among different instances (i.e., they are not biased toward easy or hard instances). This is because given an instance, all solution times are relative to the fastest solver on that instance. The simplest way to interpret performance profiles is that the highest curve on the graph corresponds to the best performing algorithm. Note that, as is common practice, all times were shifted by 1 second.

4.2 Set of Test Instances

We tested the factorization on 441 matrices arising from the `BasisLIB_INT` (available at either [Steffy 2010a] or [Steffy 2010b] and the San Jose State University Singular Matrix repositories (available at [Foster and Botev 2009] and within MATLAB via the `SJget` interface).

The BasisLIB_INT repository is a collection of 276 integral real world LP basis matrices and right hand side vectors obtained as output from the QSopt_ex solver [Applegate et al. 2007a,b]. The matrices within the BasisLIB_INT repository were used as a test bed for exact rational factorization algorithms within linear programming [Cook and Steffy 2011; Gleixner 2015; Steffy 2011]; thus are a strong test set for the SPEX Left LU factorization.

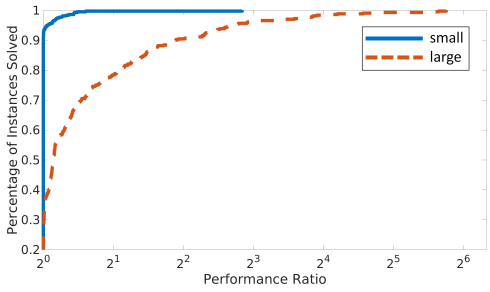
The San Jose State University Singular Matrix database contains 700 matrices arising from real world applications; these matrices are highly ill-conditioned and claimed to be numerically singular to double precision. This database of matrices was used in [Foster and Davis 2013] to evaluate the accuracy of SPQR_RANK, a library for computing accurate solutions of nearly singular linear systems, and computations on rank deficient matrices such as estimating the numerical rank and computing null space bases. It is based on a sparse QR factorization using conventional floating-point arithmetic. For nearly singular matrices, SPQR_RANK is far more accurate than the MATLAB backslash, but it is only intended for computing highly-accurate solutions, not exact ones, which is the purpose of SPEX Left LU. Of the 700 matrices within the database, 480 are square. We printed the square matrices to 16 decimal digits of precision and attempted to factorize them with SPEX Left LU. Of these matrices, SPEX Left LU determined that 165 were nonsingular, 228 were singular, and the remaining 87 could not be classified after 24 hours of run time. Henceforth, these 165 nonsingular matrices will be referred to as the SJ database. For convenience, the 165 printed to 16 decimal digits matrices can be found at [Foster and Botev 2022].

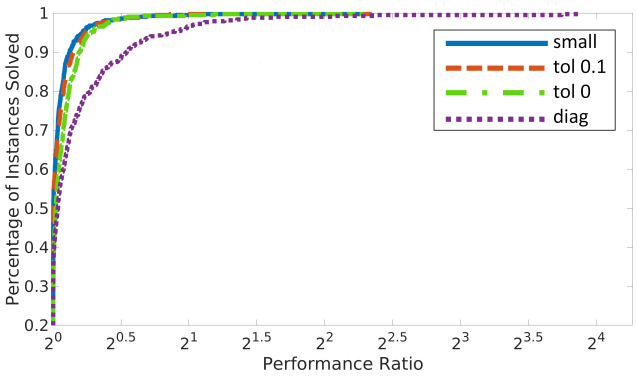
4.3 Pivoting Schemes in SPEX Left LU

Lourenco et al. [2019] used a partial pivoting scheme in which the smallest nonzero entry in each column is selected as the pivot element with the conjecture that this pivoting scheme would reduce the bit-length of entries in the ensuing submatrix (consequently reducing the factorization time). However, no extensive computational tests were performed to test this conjecture. In this section, we set all parameters to their default values and computationally compare several pivoting schemes to test their impact on the factorization time.

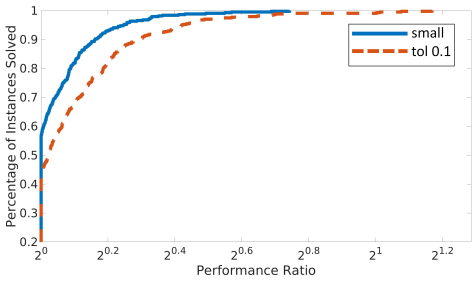
First, we test the hypothesis that small pivoting dominates large pivoting by testing these two schemes: **small**, the (first) smallest entry in column k is selected as the k^{th} pivot and **large**, the largest entry in column k is selected as the k^{th} pivot. Across the 441 instances, small pivoting lead to a faster factorization on 86% of the matrices while being approximately two times faster on average (note that large pivoting timed out on 5 instances from the SJ database; thus its run time was set to the upper bound of 24 hours). Graphically, we illustrate the clear superiority of small pivoting via the performance profile given in Figure 1. In this profile, we see that small pivoting completely dominates large pivoting.

Fig. 1. Small Pivoting Dominates Large Pivoting on All Matrices

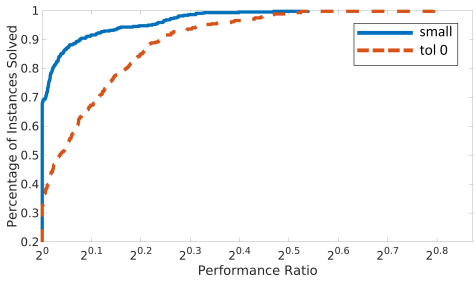




(a) Small Dominates Diagonal Pivoting



(b) Small dominates tol 0.1



(c) Small dominates tol 0

Fig. 2. Comparison of Small Pivoting Schemes

Since small pivoting dominated large pivoting, we next determined what version of small partial pivoting works best. To do so, we tested four pivoting schemes: the **small** pivoting scheme, as well as 3 additional schemes that select the diagonal element if it is within some tolerance of the smallest pivot. The three new methods select the k^{th} pivot as follows: **diag**: if it is nonzero, the diagonal entry is selected as the k^{th} pivot, otherwise the smallest entry in the column is the pivot element, **tol 0**: the diagonal entry is selected as the pivot if it has the same magnitude as the smallest entry, otherwise the smallest entry is selected, and **tol 0.1**: the diagonal entry is selected as the pivot if its magnitude is within 10% of the smallest entry, otherwise the smallest entry is selected. Numerically, the run times of small, tol 0, and tol 0.1 were very similar; thus, we compare these pivoting schemes via a sequence of performance profiles. First, we show that diag is the worst of these four pivoting schemes via Figure 2a. In this performance profile, we notice that each of small, tol 0, and tol 0.1 dominate diag pivoting across all instances. It also appears from this figure that small is the superior pivoting scheme. In order to verify this, we perform two pairwise comparisons which compare small, tol 0, and tol 0.1. Figure 2b indicates that small dominates tol 0.1 and Figure 2c indicates that small slightly outperforms tol 0; however, the magnitudes of run times difference is so slight (note the scale of the x-axis) that practically speaking these three pivoting schemes are identical on this data set.

4.4 Accuracy of MATLAB Sparse Backslash

This section utilizes SPEX Left LU to benchmark the accuracy of MATLAB sparse backslash, the state-of-the-art solver for sparse linear systems. To benchmark the MATLAB backslash, we compute the exact solution of the linear system using SPEX Left LU and report this solution to double precision. Then, we compare the solution obtained from SPEX Left LU to the solution obtained from MATLAB. Note that, for a sparse unsymmetric input matrix A, MATLAB sparse backslash utilizes the unsymmetric multifrontal package (UMFPACK) [Davis 2004] to compute its LU factorization, sometimes followed by one or more steps of iterative refinement.

Table 1 shows the forward and backward error of both methods on the `BasisLIB_INT` repository. Note that x_s refers to the exact solution obtained from SPEX Left LU. From this table, we see that our trust in the accuracy of commercial sparse solvers is mostly justified. Namely, backslash produces a nearly exact solution (i.e., error less than 10^{-12}) for over 95% of the matrices in the collection and only fails to find a suitable solution (i.e., error more than 10^{-2}) for less than 3% of the matrices. We also note that both backslash and SPEX Left LU do not meet a strong relative forward error bound ($\|Ax-b\|/\|b\|$) for approximately 5% of instances, illustrating those matrices which are so poorly conditioned, the matrix vector multiply Ax induces large roundoff error.

Table 1. Accuracy of Commercial Solvers Computing x (BasisLIB)

Threshold	Error	Method	
		SPEX Left LU	Backslash
$\leq 10^{-12}$	$\|x_s - x\|/\|x_s\|$	N/A	95.65%
	$\|Ax-b\|/\|b\|$	93.12%	92.75%
$\leq 10^{-6}$	$\|x_s - x\|/\|x_s\|$	N/A	96.74%
	$\|Ax-b\|/\|b\|$	94.57%	94.20%
$\leq 10^{-2}$	$\|x_s - x\|/\|x_s\|$	N/A	97.10%
	$\|Ax-b\|/\|b\|$	94.93%	94.57%

Likewise, Table 2 shows the accuracy of these solvers on the 165 identified nonsingular matrices from the SJ database. For these tests, we utilize a right hand side vector, b, as a vector of all ones. For this repository, we see that backslash produces a nearly exact solution (i.e., error less than 10^{-12}) for only 31% of the matrices, a decent solution (i.e., error less than 10^{-6}) for 46% of the matrices, but produces a poor solution (i.e., error more than 10^{-2}) for 28% of the matrices. These results, though dramatic, are not surprising, based on the fact that the error bound on $\|Ax-b\|/\|b\|$ for both SPEX Left LU and backslash is poor for all measured thresholds, indicating this collection is incredibly ill-conditioned. In general, one would expect error bounds more similar to those of Table 1; however, the SJ database represents a collection of matrices in which exactness is required in order to obtain solutions to linear systems which are flagged as numerically singular.

4.5 Comparison to Iterative Exact Methods

The purpose of this section is illustrate the relevance of our approach by presenting our contributions in the light of a broader context and compare our methods to the state-of-the-art in terms of both algorithmic aspects and software implementations. In addition to direct methods (which were the focus of the paper up to this section), iterative methods are a widely-used alternative to direct (e.g., factorization based) approaches to solve sparse linear systems. Iterative methods have notably been extended to exactly solve (with high-probability) sparse linear systems; two of the approaches at the forefront of these efforts make use of either Chinese remaindering or p-adic lifting [Dixon 1982] on top of either

Table 2. Accuracy of Commercial Solvers Computing x (SJ database)

Threshold	Error	Method	
		SPEX Left LU	Backslash
$\leq 10^{-12}$	$\|x_s - x\|/\|x_s\|$	N/A	30.91%
	$\|Ax - b\|/\|b\|$	11.51%	10.91%
$\leq 10^{-6}$	$\|x_s - x\|/\|x_s\|$	N/A	46.06%
	$\|Ax - b\|/\|b\|$	39.39%	45.45%
$\leq 10^{-2}$	$\|x_s - x\|/\|x_s\|$	N/A	72.12%
	$\|Ax - b\|/\|b\|$	67.72%	67.88%

Wiedemann's [Wiedemann 1986] algorithm or block Lanczos iterations [Eberly and Kaltofen 1997; Hovinen 2004; Simon 1984].

We compare our exact factorization to the exact iterative routines within LinBox [Dumas et al. 2002] for the following reasons: (1) LinBox is extremely comprehensive: it includes tools for exact linear algebra computations over integers, rational numbers, and finite fields and rings; it can solve linear systems, and compute several matrix invariants, such as minimal and characteristic polynomials, rank, determinant, Smith normal form; it can find least-norm, least-squares solutions to singular and inconsistent systems. (2) Most importantly, LinBox is primarily designed to handle sparse matrices, and it contains implementations of both the Wiedemann and Lanczos approaches over finite fields. This in contrast to other exact solvers over finite fields such as NTL [Shoup et al. 2001], Magma [Bosma et al. 1997] and FLINT [Hart 2010]. Specifically, we compare to the most current stable releases of LinBox (and its dependencies), namely: LinBox 1.6.3 (iterative algorithms) [Dumas et al. 2002], Givaro 4.1.1 (modular arithmetic) [Dumas et al. 2008a], fflas-ffpack 2.4.3 (linear algebra over finite fields) [Dumas et al. 2008b].

Altogether, as further explained below, iterative methods had some advantages; specifically in the case where the inputs were large and the iterative methods correctly solved the linear system. However, the major issue is that these routines suffered from a lack of reliability. Specifically, across the 441 instances, both Wiedemann and Lanczos failed for 37 matrices (approx 8%): 9 segmentation faults and 28 incorrect solutions (due to the 9 segmentation faults, where appropriate, the comparisons below refer to 432 instances instead of 441 instances). This issue provides evidence that these exact iterative routines within LinBox are not appropriate for exactly solving *all* sparse linear systems (it is out of the scope of this paper to determine if the issue stems from the algorithms or the implementation).

That said, with respect to the run-times, the results are in line with the literature. Specifically, SPEX Left LU outperformed LinBox for about 86% of *easy* instances (i.e., quick to solve) while LinBox outperformed SPEX Left LU for about 84% of *hard* instances (i.e., longer run times for both). Then again, LinBox produced incorrect solutions on 24% of the tested hard instances (in addition to 8 segmentation faults). Thus, though LinBox can be significantly faster on some instances, there is no guarantee that an exact solution is returned, which could be problematic for some applications.

Altogether, it is likely that the best approach for exactly solving a sparse linear system is hybrid in nature. For completeness, we give a rough sketch of such a hybrid algorithm, but ironing out the details of such approach is out of the scope of this paper. If the given instance is solved quickly by SPEX (say, within 1 minute), which comprised the majority of problems in our test case, then utilize our exact factorization. Otherwise, use Linbox and check the correctness of the returned solution. If the solution were to be incorrect, then SPEX would need to be used.

Below, Subsections 4.5.1 and 4.5.2 further analyze the accuracy results and run times, respectively.

4.5.1 Reliability. In terms of reliability and stability, we present the results for the BasisLIB and SJ Databases separately due to the fact that the SJ Database specifically contains numerically challenging matrices. In order to determine the accuracy of each algorithm, we solve the linear system $A\mathbf{x} = \mathbf{b}$ exactly using each of SPEX Left LU, Wiedemann, and Lanczos (the latter two via LinBox). Then this solution is converted to double precision and the relative errors are computed.

Across the 276 BasisLIB instances, the 2-norm error between the three algorithms was 0. This means that Lanczos, Wiedemann, and SPEX Left LU all returned the same (exact solution). Thus, using any of these three algorithms, the solution vector $\mathbf{x}$ can be obtained either exactly or to any user-specified level of precision.

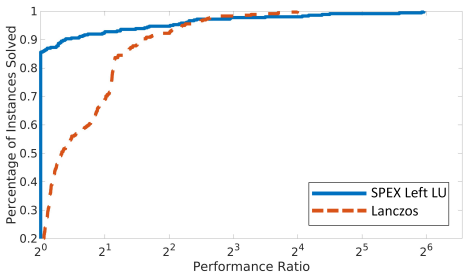
From the 165 SJ database instances, both the Wiedemann and Lanczos code suffered segmentation faults on nine of the instances. Due to these segmentation faults, where appropriate, the comparisons below refer to 156 instances instead of 165 instances. On the remaining instances, 127 were solved correctly (i.e., 2-norm error of 0) and 28 were solved incorrectly (2-norm error > 0); notably, Linbox returned no warning or error when returning an incorrect solution. Importantly, we are certain that these incorrect solutions were *not* the result of conversion errors or any other reason extraneous to the algorithms/implementation, because for both SPEX and LinBox, the input was the identical integer matrices (after scaling) and the output was formed utilizing identical GMP functions; this is evident by the fact that, when the solutions were correct, the two-norm difference was **exactly** zero.

4.5.2 Run Time. In terms of run times, we consider 432 instances (thereby excluding the 9 instances which LinBox suffered a segmentation fault). In our tests, the Lanczos algorithm nearly uniformly outperformed Wiedemann (was faster for 89% of instances with an average and geoemetric mean run time 8% and 40% smaller respectively); thus, for the sake of simplicity we restrict our comparison to the Lanczos algorithm. Across the 432 instances, SPEX Left LU exactly solved the linear system $A\mathbf{x} = \mathbf{b}$ faster than Lanczos for 74.8% of instances.

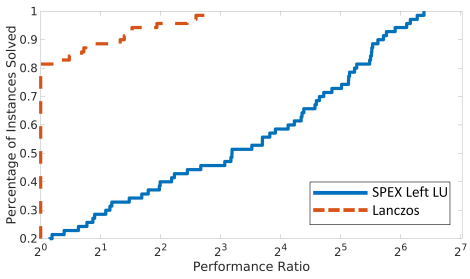
That said, the instances within BasisLIB and the SJ Database are mostly biased towards instances with small run times. Thus, to provide a more detailed analysis, we further partition the set into *easy* instances (those requiring less than 1 minute of run time by at least one of the algorithms) and *hard* instances (those requiring at least one minute of run time for both algorithms). Figure 3a gives a performance profile for the 362 easy instances showing that SPEX Left LU outperformed Lanczos for 85.6% of these matrices and Figure 3b gives a performance profile of the 70 hard instances showing that Lanczos outperformed SPEX Left LU for 84.4% of these hard matrices (that said, 8 of the 9 segmentation faults would be a part of this hard set). At first glance, these results are in line with the literature; namely direct methods are generally preferable when instances are smaller/sparser (thus suffering less fill-in) while iterative methods are generally faster on larger/denser instances. However, it is important to contextualize these results: a key caveat of Lanczos/LinBox is that it produced incorrect solutions for 17 of the 70 hard instances (in addition to 8 segmentation faults) and for 11 of the 352 easy instances (in addition to 1 segmentation fault). In short, Lanczos/LinBox failed (returned an incorrect solution or a segmentation fault) in approximately 3% of the easy instances and 32% of the hard instances. The Appendix gives a table containing the comprehensive results for all instances, along with the estimated condition number and norm of each matrix.

5 CONCLUSION

This paper presents the SPEX Left LU software package for exactly solving sparse linear systems. In addition to being submitted to the Collected Algorithms of the ACM, the SPEX Left LU package is hosted and kept up-to-date as a component of SPEX [Lourenco et al. 2022] and as a component of SuiteSparse [Davis et al. 2016].



(a) SPEX Left LU runtime outperforms Lanczos in the 362 easy instances.



(b) Lanczos runtime outperforms SPEX Left LU in 70 hard instances.

Fig. 3. SPEX Left LU vs Lanczos: It is important to contextualize the run time results. Specifically, Lanczos/LinBox returned an incorrect solution in 11 (≈3%) of the easy instances and 17 (≈24%) of the hard instances. (In addition, this results and performance profiles exclude the 9 instances (1 easy and 8 hard) where Lanczos/Linbox returned a segmentation fault.)

In order to showcase the performance of SPEX Left LU, the paper includes a robust set of computational results, from which we draw the following conclusions:

- For exact factorizations, choosing the smallest column entry as a pivot substantially outperforms choosing the largest entry. In short, choosing small pivots flattens the growth curve of the entries' sizes, and thus, improves the run-time. This is counter-intuitive as choosing large pivots is better for traditional elimination algorithms.
- The trust in commercial solvers is mostly justified as MATLAB sparse backslash provides solutions of high accuracy for the vast majority of our instances; however, exact arithmetic is necessary for highly ill-conditioned matrices and especially for matrices flagged as numerically singular.
- When comparing SPEX Left LU to exact iterative methods (Wiedemann's [Wiedemann 1986] and block Lanczos [Eberly and Kaltofen 1997; Hovinen 2004; Simon 1984], the results are in line with the literature. Specifically:
  - Direct methods, like SPEX Left LU, are generally preferable for those sparse matrices without substantial fill-in, as SPEX Left LU outperforms the iterative methods for a majority of these *easy* instances. Thus conclusion is in agreement with previous studies: (1) in the context of linear programming basis validation, Cook and Steffy [2011] state that "We found Wiedemann's method to not be attractive for our LP test instances [...] as the LU factorizations for these very sparse problems can be computed very quickly"; (2) current exact LP solvers such as QSopt [Applegate et al. 2007a] and SoPLEX [Gleixner et al. 2017; Wunderling 1997], use rational (not iterative) linear system solvers as a building block.
  - Iterative methods, like Wiedemann and Lanczos, are generally preferable for those sparse matrices in which, if factorized, substantial fill-in would occur. This is evidenced by the fact that the iterative approaches outperformed SPEX for the majority of these *hard* instances. This too is in line with the literature as Cook and Steffy [2011] states "For other classes of sparse matrices for which LU factorizations are not possible without significant fill-in, we would expect Wiedemann's method to perform more competitively."
- In terms of implementation, LinBox failed (returned an incorrect solution or a segmentation fault) in approximately 3% of the easy instances and 32% of the hard instances. Thus, our software package would be required to correctly solve these instances as well, regardless of their size, fill-in, or run time. Altogether, the best approach for exactly solving a sparse linear system is likely to be hybrid in nature, utilizing a direct solver like SPEX for those *easy*

instances while iterative methods are used for *hard* instances—though the details of such an algorithm is outside the scope of this paper.

In short, we feel confident saying that our approach and implementation is among the state-of-the-art for exactly solving sparse linear systems. Specifically, even for troublesome matrices; barring an out-of-memory condition (which is properly detected, reported, and safely handled), due to the deterministic nature of our algorithms, the solution obtained from SPEX Left LU is guaranteed to be exact.

ACKNOWLEDGMENTS

The work is partially supported by the National Science Foundation under Grant No OAC-1835499. Additionally, the first author was supported by the Texas A&M Graduate Merit Fellowship, the Texas A&M Graduate Teaching Fellowship, and the USNA Junior NARC.

REFERENCES

Patrick R Amestoy, Timothy A Davis, and Iain S Duff. 2004. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)* 30, 3 (2004), 381–388.

DL Applegate, W Cook, S Dash, and DG Espinoza. 2007a. QSopt ex. *World Wide Web. http://www. dii. uchile. cl/˜ daespino/QSoptExact_doc/main. html* (2007).

David L Applegate, William Cook, Sanjeeb Dash, and Daniel G Espinoza. 2007b. Exact solutions to linear programming problems. *Operations Research Letters* 35, 6 (2007), 693–699.

Erwin H Bareiss. 1968. SylvesterâĂŹs identity and multistep integer-preserving Gaussian elimination. *Mathematics of computation* 22, 103 (1968), 565–578.

Wieb Bosma, John Cannon, and Catherine Playoust. 1997. The Magma algebra system I: The user language. *Journal of Symbolic Computation* 24, 3-4 (1997), 235–265.

Benjamin A Burton and Melih Ozlen. 2012. Computing the crosscap number of a knot using integer programming and normal surfaces. *ACM Transactions on Mathematical Software (TOMS)* 39, 1 (2012), 4.

William Cook and Daniel E Steffy. 2011. Solving very sparse rational systems of equations. *ACM Transactions on Mathematical Software (TOMS)* 37, 4 (2011), 39.

Tim Davis, WW Hager, and IS Duff. 2014. SuiteSparse. *htt p://faculty. cse. tamu. edu/davis/suitesparse. html* (2014).

Timothy A Davis. 2004. Algorithm 832: UMFPACK V4. 3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software (TOMS)* 30, 2 (2004), 196–199.

Timothy A Davis, John R Gilbert, Stefan I Larimore, and Esmond G Ng. 2004. Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)* 30, 3 (2004), 377–380.

Timothy A Davis and Ekanathan Palamadai Natarajan. 2010. Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. *ACM Transactions on Mathematical Software (TOMS)* 37, 3 (2010), 1–17.

Timothy A Davis, Sivasankaran Rajamanickam, and Wissam M Sid-Lakhdar. 2016. A survey of direct methods for sparse linear systems. *Acta Numerica* 25 (2016), 383–566.

John D Dixon. 1982. Exact solution of linear equations using P-adic expansions. *Numer. Math.* 40, 1 (1982), 137–141.

Elizabeth D Dolan and Jorge J Moré. 2002. Benchmarking optimization software with performance profiles. *Mathematical programming* 91, 2 (2002), 201–213.

Jean-Guillaume Dumas, Thierry Gautier, Mark Giesbrecht, Pascal Giorgi, Bradford Hovinen, Erich Kaltofen, B David Saunders, Will J Turner, Gilles Villard, et al. 2002. LinBox: A generic library for exact linear algebra. In *Proceedings of the 2002 International Congress of Mathematical Software, Beijing, China*. 40–50.

Jean-Guillaume Dumas, Thierry Gautier, Pascal Giorgi, Jean-Louis Roch, and Gilles Villard. 2008a. Givaro-3.2. 13rc1: C++ library for arithmetic and algebraic computations. (2008).

Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. 2008b. Dense linear algebra over word-size prime fields: the FFLAS and FFPACK packages. *ACM Transactions on Mathematical Software (TOMS)* 35, 3 (2008), 1–42.

Wayne Eberly and Erich Kaltofen. 1997. On randomized Lanczos algorithms. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*. 176–183.

Jack Edmonds. 1967. Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards Sect. B* 71 (1967), 241–245.

Adolfo R Escobedo and Erick Moreno-Centeno. 2015. Roundoff-Error-Free Algorithms for Solving Linear Systems via Cholesky and LU Factorizations. *INFORMS Journal on Computing* 27, 4 (2015), 677–689.

Leslie Foster and NB Botev. 2009. San Jose State University singular matrix database (stored in matrix-market format in binary floating-point). (2009). http://www.math.sjsu.edu/singular/matrices

Leslie Foster and NB Botev. 2022. San Jose State University singular matrix database (stored in text to 16 decimal digits). (2022). https://github.com/clouren/SJ_16_Digits

Leslie V. Foster and Timothy A. Davis. 2013. Algorithm 933: Reliable Calculation of Numerical Rank, Null Space Bases, Pseudoinverse Solutions, and Basic Solutions Using SuitesparseQR. *ACM Trans. Math. Softw.* 40, 1, Article 7 (Oct. 2013), 23 pages. https://doi.org/10.1145/2513109.2513116

Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. 2007. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software (TOMS)* 33, 2 (2007), 13.

Bernd Gärtner. 1999. Exact arithmetic at low cost–a case study in linear programming. *Computational Geometry* 13, 2 (1999), 121–139.

John R Gilbert and Tim Peierls. 1988. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Statist. Comput.* 9, 5 (1988), 862–874.

Ambros Gleixner, Leon Eifler, Tristan Gally, Gerald Gamrath, Patrick Gemander, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Matthias Miltenberger, et al. 2017. The SCIP optimization suite 5.0. (2017).

Ambros M Gleixner. 2015. Exact and fast algorithms for mixed-integer nonlinear programming. (2015).

Gene H Golub and Charles F Van Loan. 2012. *Matrix computations*. Vol. 3. JHU Press.

Torbjrn Granlund et al. 2015. *GNU MP 6.0 Multiple Precision Arithmetic Library*. Samurai Media Limited.

Thomas C Hales. 2005. A proof of the Kepler conjecture. *Annals of mathematics* (2005), 1065–1185.

William B Hart. 2010. Fast library for number theory: an introduction. In *International Congress on Mathematical Software*. Springer, 88–91.

Nicholas J Higham. 2002. *Accuracy and Stability of Numerical Algorithms*. Vol. 80. SIAM.

Roger A Horn and Charles R Johnson. 2012. *Matrix Analysis*. Cambridge University Press.

Bradford Hovinen. 2004. *Blocked lanczos-style algorithms over small finite fields*. Ph.D. Dissertation. Citeseer.

William Kahan. 1996. IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE* 754, 94720-1776 (1996), 11.

Ed Klotz. 2014. Identification, assessment, and correction of ill-conditioning and numerical instability in linear and integer programs. In *Bridging Data and Decisions*. INFORMS, 54–108.

Hong R Lee and B David Saunders. 1995. Fraction free Gaussian elimination for sparse matrices. *Journal of symbolic computation* 19, 5 (1995), 393–402.

Joseph WH Liu. 1991. A generalized envelope method for sparse factorization by rows. *ACM Transactions on Mathematical Software (TOMS)* 17, 1 (1991), 112–129.

Christopher Lourenco, Jinhao Chen, Erick Moreno-Centeno, and Timothy Davis. 2022. SParse EXact software for solving linear systems. (2022). https://github.com/clouren/spex

Christopher Lourenco, Adolfo R Escobedo, Erick Moreno-Centeno, and Timothy A Davis. 2019. Exact Solution of Sparse Linear Systems via Left-Looking Roundoff-Error-Free LU Factorization in Time Proportional to Arithmetic Work. *SIAM J. Matrix Anal. Appl.* 40, 2 (2019), 609–638.

René M Montante-Pardo and Marco A Méndez-Cavazos. 1977. Un método númerico para cálculo matricial. *Revista Técnico-Científica de Divulgación* 2 (1977), 1–24.

Victor Shoup et al. 2001. NTL: A library for doing number theory. (2001).

Horst D Simon. 1984. The Lanczos algorithm with partial reorthogonalization. *Mathematics of computation* 42, 165 (1984), 115–142.

Dan Steffy. 2010a. BasisLIB INT. (2010). http://hpac.imag.fr/Matrices/BasisLIB/

Dan Steffy. 2010b. BasisLIB INT. (2010). https://github.com/clouren/BasisLIB_INT

Daniel E Steffy. 2011. *Topics in exact precision mathematical programming*. Ph.D. Dissertation. Georgia Institute of Technology.

Douglas Wiedemann. 1986. Solving sparse linear equations over finite fields. *IEEE transactions on information theory* 32, 1 (1986), 54–62.

Roland Wunderling. 1997. SoPlex: The sequential object-oriented simplex class library. (1997).

A COMPREHENSIVE COMPUTATIONAL RESULTS

Tables 3, 4, 5, and 6 present exhaustive computational results for large (those requiring ≥ 1 hour of run time), medium (those requiring between 1 hour and 1 minute of run time), small (those requiring between 1 minute and 0.1 seconds of run time) and tiny (those requiring less than 0.1 seconds) instances, respectively. In each table, the first five columns give the name (or index from SJ database), dimension, number of nonzeros, MATLAB estimated condition number, and MATLAB estimated sparse norm in each matrix. Column 6 gives the run time of SPEX Left LU using the selected pivoting scheme tol 0. Lastly, columns 7 and 8 give the relative run time of Wiedemann and Lanczos, respectively. Note that those instances in which these algorithms seg-fault are indicated with **SF** and those instances whose solutions were incorrect are indicated with bold red text.

						Relative Run Time	
Matrix Name	n	nnz	Cond	norm	SPEX Left LU time (hr)	Wiedemann	Lanczos
366	20640	97353	2.03E+15	7.69E+05	24.00	**SF**	**SF**
400	10964	233741	1.29E+15	1.99E+01	21.14	0.05	0.04
365	20545	85537	4.05E+21	3.38E+09	18.90	**SF**	**SF**
388	11532	44206	5.88E+12	2.90E+08	17.01	**SF**	**SF**
369	10000	49699	6.90E+20	4.31E+04	16.72	0.02	0.03
385	10672	232633	2.34E+14	1.89E+01	15.92	**SF**	**SF**
367	7337	156508	7.64E+13	1.83E+01	8.05	0.06	0.05
359	13436	71594	3.82E+15	1.41E+04	7.75	**SF**	**SF**
695	14454	147972	3.10E+12	5.31E+03	7.74	0.12	0.11
696	14454	147972	2.96E+12	5.31E+03	7.72	0.11	0.11
368	7337	154660	1.87E+24	1.83E+01	7.38	0.06	0.05
350	6774	33744	7.68E+13	1.26E+06	7.13	0.02	0.02
352	5773	71701	8.73E+12	1.29E+08	5.42	0.04	0.04
336	5005	20033	6.90E+16	6.76E+06	5.35	0.02	0.01
88	4875	315891	3.70E+16	3.47E-10	5.31	***0.05**	***0.04**
344	3363	99471	4.08E+13	3.79E+09	5.07	0.04	0.03
119	3251	65875	2.62E+16	1.27E+07	4.85	0.02	0.02
213	3402	130371	2.67E+16	3.98E+14	4.70	0.03	0.03
117	3973	79077	2.94E+19	1.28E+02	4.66	0.03	0.03
337	5321	65693	2.10E+13	4.68E+06	4.66	0.03	0.03
346	7055	30082	1.65E+12	4.02E+02	4.38	**SF**	**SF**
347	7055	30082	6.34E+17	4.02E+02	4.24	***0.03**	***0.03**
140	3937	25407	1.04E+17	3.27E+11	3.88	0.02	0.02
120	2163	74464	4.36E+16	8.16E+01	3.49	0.02	0.01
142	3937	25407	1.04E+17	3.27E+11	3.45	0.02	0.02
222	4257	37465	2.52E+16	1.46E+02	3.21	***0.03**	***0.02**
221	4257	37465	2.52E+16	1.46E+02	3.21	***0.02**	***0.02**
364	13935	63307	2.35E+18	1.26E+05	3.20	**SF**	**SF**
223	4257	37465	1.74E+19	1.46E+02	3.10	***0.02**	***0.02**

						Relative Run Time	
Matrix Name	n	nnz	Cond	norm	SPEX Left LU time (hr)	Wiedemann	Lanczos
341	2880	18229	9.74E+13	1.39E+04	2.62	0.01	0.01
349	4101	82682	1.44E+13	1.34E+01	2.60	0.06	0.05
130	2880	18229	5.98E+18	1.33E+07	2.25	0.01	0.01
138	4101	81057	1.80E+24	1.34E+01	2.17	0.06	0.05
pilot87.pre	1540	30916	Inf	Inf	1.70	0.02	0.02
118	2568	75628	4.01E+15	1.26E+10	1.57	0.07	0.06
pilot87	1625	31396	Inf	Inf	1.14	0.02	0.02
156	4800	102252	3.46E+61	3.63E+05	1.06	0.13	0.12

Table 3. Comprehensive Results: Large Instances

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (min)	Wiedemann	Lanczos
217	4720	20042	3.51E+48	1.89E+02	52.61	0.08	0.08
gen4	375	8919	8.43E+236	Inf	47.79	0.02	0.02
gen4.pre	367	9322	Inf	Inf	43.30	0.02	0.02
self	924	157411	1.20E+07	1.57E+11	41.18	0.03	0.03
slptsk	2315	34430	4.35E+185	NaN	35.48	0.27	0.29
260	1000	1000000	1.15E+14	5.36E-01	33.21	***0.49**	***0.47**
335	6747	29195	2.22E+18	5.41E+08	33.12	0.23	0.21
gen1	329	11016	Inf	Inf	28.96	0.04	0.04
57	3008	20698	2.37E+28	8.44E+09	19.43	0.09	0.09
207	1919	32399	2.10E+18	2.92E+00	19.40	0.06	0.07
155	3200	68026	1.78E+47	1.61E+05	19.16	0.20	0.18
55	3008	20715	1.94E+27	7.90E+08	18.24	0.08	0.08
87	2500	12349	4.35E+17	9.83E+03	17.91	0.07	0.07
154	1280	47906	9.87E+24	7.94E+04	16.28	0.06	0.05
122	1651	49062	8.22E+27	4.49E+02	13.93	0.13	0.11
329	5308	22680	1.67E+14	2.99E+06	11.09	0.52	0.45
330	5308	22592	1.66E+14	2.99E+06	11.08	0.51	0.44
159	1050	26198	9.00E+15	2.10E+07	10.58	0.06	0.06
pla8_sig185	39835	196256	4.29E+09	1.04E+03	10.43	0.27	0.25
355	1409	42760	2.28E+13	2.67E+05	9.31	0.18	0.16
pilot	1132	16624	1.12E+175	NaN	6.88	0.05	0.05
320	1733	22189	1.20E+13	1.18E+11	6.02	0.25	0.25
maros-r7	1350	31923	7.43E+06	3.99E+10	5.89	0.05	0.04
236	1000	1000000	1.46E+19	3.23E+00	5.49	***3.00**	***2.88**
340	8765	42471	6.15E+14	1.00E+15	5.10	**SF**	**SF**

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (min)	Wiedemann	Lanczos
jendrec1	1779	34196	9.54E+217	NaN	5.08	2.76	2.66
stat96v4	3139	23752	9.62E+17	1.63E+21	4.85	0.23	0.22
58	3083	11767	2.30E+21	1.21E+09	4.78	0.34	0.31
pla85900.nov21	40304	230558	5.46E+08	9.76E+02	4.77	0.67	0.65
momentum3	3254	15159	Inf	Inf	4.25	0.38	0.36
153	765	24382	1.67E+14	1.35E+17	4.20	0.14	0.13
232	1000	1000000	4.50E+21	8.11E-01	4.02	***4.16**	***3.84**
54	3268	20712	1.27E+27	2.50E+12	3.26	0.58	0.55
240	1000	1000000	1.44E+22	6.46E+00	3.11	***5.47**	***5.46**
296	1258	7682	1.03E+13	2.05E+07	2.68	0.13	0.12
256	1000	1000000	1.06E+23	2.99E+00	2.65	***6.45**	***6.05**
56	3268	20963	1.27E+27	2.50E+12	2.54	0.86	0.76
cont11_l	58936	179556	3.65E+26	7.37E+00	2.46	0.01	0.01
264	1000	1000000	8.23E+21	4.47E-01	2.19	***7.11**	***6.86**
mod2.pre	4422	12914	3.72E+246	NaN	2.18	1.48	1.40
brd14051	16360	180847	2.96E+08	2.74E+02	2.14	0.60	0.58
world	4261	12190	5.65E+241	NaN	2.10	1.30	1.23
157	4800	27520	1.03E+14	2.20E+00	2.05	1.75	1.61
fome13	24884	70839	7.99E+15	4.02E+11	1.94	0.58	0.53
121	1159	11047	1.57E+19	1.17E+02	1.85	0.21	0.20
mod2	4435	12985	1.38E+223	NaN	1.78	1.74	1.65
160	1374	8588	4.11E+15	1.10E+03	1.69	0.24	0.24
309	2837	10967	5.85E+12	7.41E+05	1.64	0.97	0.90
314	2836	10965	5.85E+12	7.41E+05	1.64	0.96	0.88
l30	2492	12653	1.73E+08	2.61E+09	1.56	0.33	0.30
259	500	250000	1.15E+14	5.36E-01	1.24	***1.71**	***1.69**
scfxm1-2r-256	11812	44985	3.12E+12	2.84E+08	1.15	2.95	2.65
332	4101	36879	6.70E+20	2.02E+06	1.11	2.93	2.53

Table 4. Comprehensive Results: Medium Instances

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (s)	Wiedemann	Lanczos
gen2	328	8894	1.84E+11	4.51E+18	54.10	0.10	0.07
310	3200	18316	2.02E+13	2.20E+00	48.00	2.24	2.07
128	760	5739	1.12E+16	3.10E+08	43.91	0.19	0.18
129	760	5816	9.93E+19	3.10E+08	42.15	0.19	0.18
nemswrld	2205	13323	NaN	Inf	39.39	0.54	0.49

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (s)	Wiedemann	Lanczos
210	1484	6110	5.57E+17	1.26E+16	31.32	0.87	0.81
cont4	2802	11862	3.85E+05	3.98E+03	30.32	0.51	0.48
nug30	14681	45627	1.06E+06	7.88E+00	30.07	0.88	0.79
nug20	7733	31455	8.84E+07	7.26E+00	28.82	0.40	0.34
rat5	902	12026	4.19E+06	7.48E+05	26.66	0.17	0.16
stat96v1	5013	20325	6.35E+32	8.22E+20	25.24	0.66	0.62
51	1813	11246	3.82E+14	1.22E+00	20.19	***1.63**	***1.52**
stat96v2	12928	48009	1.06E+13	1.32E+10	20.04	1.87	1.79
nug15	5486	24736	2.41E+07	7.73E+00	19.96	0.31	0.28
141	511	2796	6.37E+15	4.21E+10	19.26	0.20	0.18
pilot.ja	567	3781	Inf	Inf	19.14	0.43	0.39
231	500	250000	8.37E+20	8.11E-01	18.84	***6.80**	***6.79**
144	511	2796	6.37E+15	4.21E+10	18.83	0.22	0.18
235	500	250000	4.74E+18	3.23E+00	18.57	***7.28**	***7.10**
pla33810	18940	123445	1.05E+08	2.20E+02	17.15	2.94	2.69
stat96v3	13485	49917	3.93E+12	6.98E+09	16.99	2.21	2.07
model10	1341	6403	1.71E+164	Inf	15.92	0.90	0.77
d2q06c	1047	5717	1.82E+188	Inf	14.96	0.73	0.63
239	500	250000	4.54E+20	6.46E+00	14.81	***9.17**	***9.20**
d15112	9197	47335	2.33E+18	2.25E+11	14.53	3.10	2.92
watson_1	5729	14544	7.98E+58	9.72E+55	13.98	6.80	6.40
315	2053	18447	6.71E+16	2.03E+04	13.37	3.73	3.44
rat7a	641	10542	1.38E+20	5.06E+05	13.18	0.20	0.18
progas	1167	6500	8.15E+103	6.78E+100	12.63	1.19	1.09
scfxm1-2b-64	5966	22682	1.30E+12	2.84E+08	12.57	4.12	3.82
255	500	250000	1.07E+22	2.99E+00	12.47	***10.58**	***10.36**
scfxm1-2r-128	5671	21943	1.64E+12	2.84E+08	12.27	3.96	3.73
stat96v5	812	3795	1.42E+63	3.93E+64	12.08	1.33	1.20
qap12	2740	12014	2.46E+07	6.84E+00	11.79	0.14	0.12
124	1220	5892	2.24E+34	2.87E+00	10.16	1.66	1.53
NSR8K	5387	46157	2.64E+07	2.06E+03	9.91	0.93	0.81
watson_1.pre	4642	12991	3.44E+58	9.72E+55	9.83	7.13	6.72
co9	2287	13481	2.32E+49	1.56E+42	9.68	1.58	1.42
301	1650	7419	5.63E+12	5.15E+02	9.65	2.84	2.58
l30.pre	1199	6030	1.79E+06	2.61E+09	8.49	0.49	0.46
127	1220	5855	6.25E+14	3.70E+02	8.48	1.97	1.74
291	1220	5860	1.80E+13	8.78E+00	8.35	1.98	1.85
126	1220	5884	1.06E+14	9.34E+00	8.33	2.00	1.78

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (s)	Wiedemann	Lanczos
293	1220	5892	1.58E+13	8.37E+00	8.21	2.05	1.91
newman2	468	7917	1.39E+225	Inf	8.10	0.39	0.37
125	1220	5892	1.58E+17	2.88E+00	8.08	2.07	1.82
289	1220	5884	1.42E+13	4.95E+00	7.85	2.13	1.99
292	1220	5888	3.45E+13	2.02E+01	7.84	2.11	1.95
287	1220	5888	4.14E+13	1.63E+01	7.60	2.17	1.93
288	1220	5852	2.74E+13	1.23E+01	7.60	2.17	2.02
stormg2_1000.pre	13926	32547	2.13E+11	2.59E+05	7.43	12.82	12.77
212	882	3354	7.98E+16	6.69E+12	7.20	1.33	1.12
209	415	2779	8.19E+17	6.47E+00	7.14	0.41	0.34
294	1220	5892	2.74E+13	1.23E+01	7.12	2.54	2.63
158	416	8562	2.42E+25	2.54E+03	7.02	0.60	0.57
263	500	250000	2.57E+21	4.47E-01	6.97	***17.74**	***18.10**
momentum2	2113	6516	2.52E+39	1.88E+37	6.95	1.61	1.40
pilotnov	549	3337	2.82E+264	NaN	6.53	0.95	0.80
nug12	2736	12037	2.25E+07	6.97E+00	6.48	0.32	0.22
stormg2_1000	14075	32597	3.77E+09	1.05E+04	6.46	15.44	14.78
295	3562	3562	1.81E+13	1.02E+06	6.23	17.41	15.82
perold	440	2584	4.45E+256	NaN	6.16	0.84	0.72
scfxm1-2r-96	4504	17205	1.35E+12	2.84E+08	6.03	4.92	4.51
53	1089	4144	6.05E+14	2.91E+04	5.77	2.02	1.95
dbic1	4795	23403	1.04E+11	4.53E+06	5.59	1.75	1.47
pilot4	289	2805	Inf	Inf	5.19	0.87	0.81
211	768	2934	1.29E+17	1.36E+13	4.95	1.53	1.27
cont1_l	1070	4649	1.22E+09	2.51E+05	4.55	0.88	0.79
model11	2039	7606	1.40E+32	3.38E+29	4.12	1.82	1.56
pcb3038	3588	46560	2.00E+06	1.55E+02	3.98	1.22	1.17
nemspmm2	949	6478	2.61E+222	Inf	3.72	1.55	1.36
fome12	12652	35969	8.93E+05	4.65E+01	2.69	5.44	4.74
nl	890	2919	5.83E+293	Inf	2.62	4.09	3.34
scfxm1-2r-64	1870	11122	8.23E+11	2.84E+08	2.28	5.91	5.35
fnl4461	5044	46977	4.12E+06	2.35E+02	2.27	3.26	2.87
152	180	2659	3.59E+17	2.04E+19	2.09	0.66	0.50
d18512	10815	55880	8.61E+05	2.09E+02	1.77	9.57	8.41
rl11849	6769	40885	1.21E+08	1.92E+02	1.75	5.34	4.65
nemspmm1	982	5023	4.94E+23	1.59E+20	1.75	2.20	2.01
co5	928	6173	3.86E+185	Inf	1.69	3.78	3.46
pla7397	5059	42683	8.55E+06	3.36E+02	1.58	4.01	3.57

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (s)	Wiedemann	Lanczos
model6	790	3425	5.17E+63	4.11E+61	1.56	2.30	2.12
258	200	40000	1.84E+14	5.36E-01	1.54	***4.96**	***4.49**
rail4284	2463	11802	3.58E+04	7.23E+00	1.45	0.88	0.74
pilot.we	554	2367	5.11E+137	4.03E+133	1.43	3.02	2.73
342	10001	49999	4.17E+18	5.00E+03	1.39	**SF**	**SF**
qap10	1510	6381	4.45E+06	6.13E+00	1.30	0.45	0.29
cq9	1187	5786	4.94E+165	Inf	1.30	4.77	4.31
ge	1675	4758	3.29E+130	7.56E+127	1.26	8.59	7.28
de080285	368	1493	1.06E+78	1.36E+79	1.24	2.08	1.55
dano3mip.pre	1091	5239	3.46E+10	1.36E+05	1.05	0.87	0.55
dano3mip	1135	5390	2.47E+10	1.36E+05	0.99	0.78	0.60
238	200	40000	3.00E+20	6.46E+00	0.94	7.94	7.79
newman	334	2156	6.20E+233	Inf	0.93	1.82	1.44
230	200	40000	5.72E+19	8.11E-01	0.86	***8.79**	***8.79**
rat1	452	2893	3.93E+27	2.26E+30	0.85	1.15	0.89
254	200	40000	1.83E+20	2.99E+00	0.81	***10.55**	***9.39**
t0331-4l	520	5034	2.75E+04	1.10E+01	0.77	0.72	0.35
rl5915	3853	28829	3.43E+06	1.03E+02	0.77	4.62	4.65
nesm	279	895	1.01E+127	1.12E+125	0.68	4.43	3.21
grow22	434	4711	2.09E+17	4.39E+16	0.68	2.28	1.98
fome11	6226	17749	6.31E+05	5.09E+01	0.65	5.73	5.77
234	200	40000	3.18E+18	3.23E+00	0.64	***11.86**	***11.54**
model9	902	4361	3.90E+23	6.53E+20	0.63	4.69	3.28
model7	646	2850	2.89E+137	2.87E+134	0.61	3.98	3.20
rl5934	3773	23917	2.94E+06	1.11E+02	0.60	4.77	4.48
model5	492	2247	7.35E+109	1.03E+108	0.57	4.24	3.46
orna1	810	2842	2.66E+13	2.55E+15	0.57	4.06	3.45
lp22.pre	1811	13146	3.89E+05	1.27E+01	0.56	1.93	1.77
132	216	812	8.10E+14	1.39E+00	0.55	1.35	0.94
179	430	1544	7.36E+26	2.00E+06	0.54	4.49	4.05
169	430	1544	6.57E+15	2.00E+06	0.54	4.37	4.14
168	430	1544	3.20E+15	2.00E+06	0.53	4.48	4.20
167	430	1544	1.91E+16	2.00E+06	0.53	4.55	4.16
164	430	1544	3.56E+16	2.00E+06	0.53	5.52	4.24
163	430	1544	1.90E+20	2.00E+06	0.53	4.78	4.23
176	430	1544	1.00E+15	2.00E+06	0.53	5.03	4.23
siena1	1265	11573	6.40E+05	7.85E+02	0.52	2.12	1.03
model4	409	1898	9.08E+213	NaN	0.52	3.78	3.11

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (s)	Wiedemann	Lanczos
165	430	1544	2.49E+14	2.00E+06	0.52	4.60	4.29
166	430	1544	2.22E+14	2.00E+06	0.51	4.66	4.45
188	430	1544	5.30E+18	2.00E+06	0.51	5.05	4.37
177	430	1544	3.78E+26	2.00E+06	0.51	5.16	4.40
194	430	1544	3.20E+19	2.00E+06	0.51	5.07	4.54
180	430	1544	9.05E+19	2.00E+06	0.51	4.92	4.45
184	430	1544	2.88E+17	2.00E+06	0.50	4.97	4.42
182	430	1544	2.88E+17	2.00E+06	0.50	5.06	4.41
186	430	1544	2.88E+17	2.00E+06	0.50	4.87	4.44
187	430	1544	2.88E+17	2.00E+06	0.50	5.04	4.43
196	430	1544	8.14E+19	2.00E+06	0.50	5.14	4.47
185	430	1544	2.88E+17	2.00E+06	0.50	4.87	4.47
202	430	1544	8.00E+19	2.00E+06	0.50	5.12	4.44
204	430	1544	7.96E+19	2.00E+06	0.50	5.01	4.50
205	430	1544	8.08E+19	2.00E+06	0.49	4.84	4.48
197	430	1544	1.15E+20	2.00E+06	0.49	5.06	4.51
203	430	1544	7.92E+19	2.00E+06	0.49	5.04	4.57
175	430	1544	4.81E+14	2.00E+06	0.49	5.04	4.55
189	430	1544	2.34E+20	2.00E+06	0.49	5.20	4.54
181	430	1544	2.88E+17	2.00E+06	0.49	5.30	4.69
de063155	313	1233	1.09E+95	6.12E+69	0.48	4.07	3.18
190	430	1544	2.31E+20	2.00E+06	0.48	5.15	4.61
192	430	1544	4.12E+20	2.00E+06	0.48	5.31	4.62
193	430	1544	2.35E+20	2.00E+06	0.48	5.28	4.62
191	430	1544	4.43E+20	2.00E+06	0.48	5.25	4.59
195	430	1544	4.66E+20	2.00E+06	0.48	5.30	4.79
172	430	1544	2.28E+19	2.00E+06	0.48	5.15	4.62
198	430	1544	8.24E+19	2.00E+06	0.48	5.37	4.68
183	430	1544	2.88E+17	2.00E+06	0.48	5.36	4.74
170	430	1544	2.93E+14	2.00E+06	0.48	5.15	4.67
199	430	1544	8.03E+19	2.00E+06	0.47	5.36	4.73
200	430	1544	7.86E+19	2.00E+06	0.47	5.23	4.71
201	430	1544	8.03E+19	2.00E+06	0.47	5.21	4.71
173	430	1544	7.05E+19	2.00E+06	0.46	5.31	4.82
171	430	1544	6.24E+15	2.00E+06	0.46	5.77	4.86
280	430	1544	2.91E+13	2.00E+06	0.46	5.51	4.57
178	430	1544	2.17E+26	2.00E+06	0.46	5.61	4.93
174	430	1544	2.33E+14	2.00E+06	0.46	5.36	4.91

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (s)	Wiedemann	Lanczos
van	7417	21681	4.24E+04	7.23E+00	0.45	9.41	7.86
scfxm1-2r-32	1447	5658	2.93E+11	2.84E+08	0.45	8.52	7.46
lp22	1796	13076	1.32E+05	1.29E+01	0.42	3.11	2.45
281	430	1544	8.60E+13	2.00E+06	0.42	5.78	4.99
206	430	1544	8.02E+19	2.00E+06	0.41	6.00	5.44
161	261	1500	1.17E+15	1.04E+03	0.39	3.88	2.29
262	200	40000	6.36E+20	4.47E-01	0.38	***19.86**	***18.88**
stocfor3	1782	4562	2.03E+31	3.50E+27	0.37	20.56	17.77
arki001	160	893	4.46E+299	Inf	0.34	6.07	5.10
momentum1	932	2792	1.18E+146	2.49E+141	0.34	9.83	8.78
usa13509	3595	19919	2.89E+08	1.70E+02	0.34	10.07	8.84
large000	823	2282	1.39E+31	1.02E+34	0.33	14.58	13.28
dfl001	3271	9276	4.54E+05	4.33E+01	0.28	4.20	3.48
277	183	1069	2.69E+13	1.15E+09	0.28	***2.47**	***1.44**
complex	327	10738	9.05E+04	5.39E+01	0.26	1.54	0.70
grow15	297	3614	1.65E+13	4.24E+16	0.25	4.03	2.82
de063157	282	1102	1.10E+97	6.45E+77	0.24	7.11	5.34
t1717	549	3657	1.89E+04	7.40E+00	0.24	1.55	0.48
greenbeb	713	3278	1.17E+25	9.16E+21	0.24	7.42	6.31
dfl001.pre	2097	6501	4.10E+05	3.99E+01	0.24	2.86	2.02
scfxm1-2r-27	1222	4753	5.08E+10	2.84E+08	0.23	13.51	10.82
ulevimin	697	1879	8.89E+103	5.41E+96	0.22	8.97	7.26
pcb3000	3058	27446	5.01E+04	6.24E+01	0.21	7.81	7.44
stair	324	3431	1.27E+19	4.02E+15	0.19	2.99	1.98
newman3	369	3662	1.58E+23	3.91E+19	0.18	4.03	3.11
stp3d	10642	25936	4.12E+05	4.28E+00	0.18	36.40	31.20
nemsemm2	789	2440	1.78E+124	1.79E+123	0.18	17.00	14.18
trento1	1070	10010	1.54E+05	6.57E+02	0.18	2.77	1.51
cr42	304	608	1.52E+35	1.86E+84	0.17	24.88	23.23
car4	122	4384	1.30E+14	4.51E+15	0.15	4.24	2.45
nug08	732	3004	2.49E+05	5.71E+00	0.14	2.66	0.46
greenbea	664	2706	2.17E+38	2.38E+32	0.14	9.91	8.49
cq5	570	2615	1.05E+164	Inf	0.13	12.16	9.62
dc1l	851	5171	1.68E+05	9.70E+02	0.13	4.29	1.30
pldd000b	537	1448	9.92E+18	1.27E+22	0.12	14.54	10.57
watson_2	1011	2703	7.02E+42	1.78E+40	0.11	19.40	16.26
25fv47	416	2061	1.42E+35	7.33E+32	0.11	7.98	5.00
237	100	10000	1.26E+19	6.46E+00	0.10	12.46	9.43

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (s)	Wiedemann	Lanczos
gran	284	1958	7.38E+32	1.25E+27	0.10	9.30	6.60
delf000	593	1606	1.27E+18	1.44E+20	0.10	18.79	16.49
ds	647	12193	1.01E+04	2.08E+01	0.10	5.62	3.50

Table 5. Comprehensive Results: Small Instances

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (ms)	Wiedemann	Lanczos
scfxm1-2r-16	752	2962	1.94E+10	2.84E+08	9.41	14.26	10.81
dolom1	806	5681	3.47E+05	2.27E+03	9.25	7.08	1.43
dc1c	808	4698	1.26E+06	9.63E+02	9.05	5.48	1.55
dg012142	892	3627	9.32E+07	2.13E+04	9.02	9.05	5.38
scfxm1-2b-16	784	2975	3.37E+10	2.84E+08	9.00	14.52	11.49
279	261	2319	7.54E+16	2.03E+04	8.81	13.00	10.74
233	100	10000	2.98E+18	3.23E+00	8.73	12.51	11.48
257	100	10000	2.48E+14	5.36E-01	8.36	***13.03**	***11.51**
pf2177	406	1772	9.40E+03	4.65E+00	8.25	4.19	0.53
253	100	10000	3.95E+19	2.99E+00	8.18	***15.20**	***11.84**
229	100	10000	6.28E+19	8.11E-01	8.10	***13.69**	***11.49**
143	131	536	1.49E+15	9.77E+09	7.06	7.22	2.73
261	100	10000	1.04E+20	4.47E-01	6.12	***19.19**	***14.65**
ch	393	1304	4.36E+110	3.12E+106	6.05	21.43	17.07
aa01	630	4187	1.29E+04	8.67E+00	5.96	7.46	1.42
139	131	536	1.49E+15	9.77E+09	5.63	8.76	3.43
air04	630	4187	1.29E+04	8.67E+00	5.48	6.20	1.56
stormg2-125	1886	4372	2.63E+07	2.26E+03	5.41	41.10	36.68
model3	310	1417	3.11E+95	6.44E+93	5.33	11.84	7.72
stormg2-125.pre	1780	4138	1.96E+09	5.43E+04	4.91	45.32	36.03
msc98-ip	2897	10006	1.63E+07	4.26E+02	4.72	41.80	37.10
df2177	414	1825	4.61E+03	4.74E+00	4.32	6.32	3.06
nug07	450	1780	5.45E+04	5.28E+00	4.09	4.86	3.13
biella1	813	5726	2.72E+04	1.18E+01	3.83	11.68	4.52
pcb1000	1156	9955	2.28E+04	4.37E+01	3.62	9.28	6.30
aa03	562	3420	1.07E+04	9.00E+00	3.56	9.70	1.42
protfold	574	2562	8.36E+03	6.50E+00	3.42	10.66	1.67
pds-100	8377	17555	1.79E+04	5.48E+00	3.38	121.91	114.99
rosen10	989	6916	2.01E+03	1.35E+03	3.34	40.93	17.40
lpl1	2692	7211	8.18E+08	1.02E+04	3.31	76.13	30.93

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (ms)	Wiedemann	Lanczos
grow7	138	1744	8.45E+12	3.53E+16	3.29	11.67	5.16
air06	562	3420	1.07E+04	9.00E+00	3.13	7.97	1.60
pds-80	9225	19432	1.12E+04	5.66E+00	2.88	195.69	166.12
pds-90	7914	16673	1.48E+04	4.13E+00	2.84	144.91	129.57
pds-70	7822	16545	1.11E+04	5.84E+00	2.76	155.28	134.17
model2	149	757	1.15E+112	4.60E+108	2.74	27.60	6.74
l9	241	1381	7.65E+21	1.74E+19	2.68	14.93	7.46
pds-60	7586	16067	8.18E+03	4.48E+00	2.61	133.01	122.99
degen3	744	5431	1.66E+04	2.00E+01	2.48	14.50	3.15
bg512142	560	2140	6.92E+06	1.95E+03	2.21	21.17	8.41
scfxm1-2r-8	403	1608	7.45E+09	2.84E+08	2.01	24.88	14.95
bas1lp	502	6651	9.77E+04	9.80E+01	1.95	23.29	6.54
gosh	379	1379	8.05E+27	4.51E+24	1.90	38.08	20.26
pilot4i	134	1220	1.50E+33	4.19E+28	1.88	20.85	8.11
rosen2	431	4143	7.85E+02	3.19E+02	1.86	46.94	9.46
pds-50	5962	12592	1.04E+04	3.49E+00	1.83	100.82	95.64
rail507	413	2005	5.39E+03	6.85E+00	1.48	21.31	7.85
scsd8	247	655	9.59E+18	1.58E+17	1.40	32.29	16.27
air05	323	1789	2.79E+04	6.94E+00	1.40	13.34	3.16
30_70_4.5_0.95_100	2754	8381	6.12E+03	4.34E+00	1.38	70.03	54.02
d6cube	223	1424	9.56E+05	4.24E+02	1.38	4.18	8.83
mitre	801	2466	4.18E+05	2.81E+03	1.28	71.92	37.51
fome21	3291	7240	9.21E+03	3.18E+00	1.24	70.06	50.55
modszk1	263	765	1.52E+21	1.25E+20	1.23	92.27	15.12
10teams	177	885	1.12E+03	5.63E+00	1.17	9.16	4.62
pds-40	4028	8478	6.64E+03	3.02E+00	1.14	84.03	54.20
fast0507	401	1908	3.72E+03	6.99E+00	1.13	22.27	5.28
south31	112	460	1.68E+265	Inf	1.10	90.21	12.86
qiulp	603	1717	8.25E+10	1.31E+09	1.09	32.35	16.01
qiu	603	1717	5.99E+10	1.31E+09	1.09	55.15	16.11
ganges	344	1123	4.19E+26	3.03E+25	1.09	47.32	23.96
cycle	284	878	1.46E+79	3.19E+75	1.06	47.12	19.64
maros	289	1143	2.76E+14	2.54E+11	0.97	39.30	11.16
30_70_4.5_0.95_98	2451	7364	7.84E+03	4.35E+00	0.94	90.63	61.09
scagr7-2r-864	680	1697	6.99E+06	3.42E+03	0.84	87.19	44.41
p05	919	2717	4.06E+04	3.06E+01	0.82	58.18	14.21
rentacar	327	1080	3.64E+05	1.01E+03	0.81	38.50	49.30
bnl2	459	1488	1.22E+15	1.10E+12	0.81	87.01	32.92

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (ms)	Wiedemann	Lanczos
30_70_4.5_0.5_100	2098	6197	1.98E+03	4.28E+00	0.75	88.59	54.45
pds-30	2643	5641	3.27E+03	3.04E+00	0.74	95.40	51.03
bandm	122	609	1.12E+27	1.05E+26	0.74	45.42	9.74
r05	919	2717	4.06E+04	3.06E+01	0.73	39.90	16.78
fome20	1718	3811	5.39E+03	3.14E+00	0.67	68.02	29.74
scfxm1-2b-4	233	965	5.92E+09	2.84E+08	0.66	49.39	14.98
scfxm1-2r-4	233	965	5.92E+09	2.84E+08	0.66	87.60	14.92
scfxm1-2c-4	233	965	5.92E+09	2.84E+08	0.66	51.71	15.03
baxter.pre	470	1274	3.55E+94	8.48E+88	0.64	94.20	41.38
nug06	267	1007	1.04E+04	4.81E+00	0.57	8.39	10.57
neos	2342	5098	3.29E+04	5.88E+00	0.56	110.87	46.57
capri	138	507	3.72E+68	3.01E+65	0.54	79.61	23.11
rail582	384	1387	4.55E+03	5.10E+00	0.53	22.83	22.75
danoint	196	790	2.11E+07	6.31E+03	0.53	45.87	5.47
neos.pre	2080	4578	1.60E+04	5.86E+00	0.48	161.55	45.24
p010	839	2486	3.96E+04	3.06E+01	0.48	81.81	21.01
rosen1	217	2528	1.23E+03	5.80E+02	0.48	65.13	11.27
rosen8	264	1850	2.53E+02	8.87E+01	0.47	88.81	10.78
seymour	537	1881	5.24E+03	5.81E+00	0.44	77.87	8.98
bnl1	223	824	3.34E+23	2.35E+20	0.44	135.85	25.00
mzzv11	1098	3189	8.92E+05	2.46E+02	0.42	83.45	39.75
scfxm3	262	1005	1.67E+17	7.58E+13	0.42	85.35	29.35
scrs8-2r-512	992	1984	1.92E+01	9.13E+00	0.38	114.47	51.21
rail516	268	936	1.31E+03	5.64E+00	0.37	15.38	20.34
sp97ar	271	2400	2.40E+04	3.93E+01	0.37	126.44	37.75
neos7	590	1434	1.46E+08	1.00E+06	0.34	234.47	53.01
stocfor2	224	576	3.54E+26	1.50E+24	0.33	174.61	36.69
dbir1	154	845	4.54E+06	1.42E+06	0.32	180.07	24.27
small000	140	383	6.77E+19	5.59E+23	0.32	122.40	31.55
neos6	174	1580	3.33E+03	7.20E+01	0.32	25.32	30.72
sp98ar	223	1782	1.69E+04	4.43E+01	0.31	59.23	62.01
woodw	168	589	1.44E+12	3.80E+10	0.29	282.23	27.99
80bau3b	154	396	3.31E+46	1.75E+45	0.29	146.51	35.63
manna81	1392	2784	3.00E+00	2.00E+00	0.29	119.16	34.38
roll3000	177	1101	2.19E+06	1.47E+03	0.28	47.75	23.19
disctom	192	565	5.92E+02	3.41E+00	0.28	11.65	14.21
dbir2.pre	281	1879	5.86E+06	1.10E+05	0.28	151.55	17.77
scfxm2	178	658	1.67E+17	7.58E+13	0.27	87.07	20.94

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (ms)	Wiedemann	Lanczos
neos11	365	1116	3.06E+03	5.06E+00	0.26	40.09	64.46
mzzv42z	787	2124	5.68E+05	2.01E+02	0.26	103.11	33.88
route	339	1290	7.27E+08	2.03E+07	0.26	95.47	38.07
degen2	217	1138	2.58E+03	1.15E+01	0.25	11.23	31.76
dsbmip	220	568	3.06E+46	2.19E+41	0.24	191.73	31.08
nsct1	120	595	4.25E+06	7.67E+05	0.23	257.58	270.62
stormg2-27	449	1019	3.67E+06	1.77E+03	0.23	311.75	40.18
baxter	256	697	3.06E+21	6.81E+15	0.23	201.24	30.85
neos1	309	944	8.24E+02	4.83E+00	0.22	13.85	9.92
crew1	127	861	7.82E+02	8.18E+00	0.22	3.32	4.40
blp-ar98	148	876	3.94E+03	9.40E+01	0.18	33.94	77.04
nsct2.pre	156	1140	3.84E+06	3.22E+05	0.16	106.01	127.32
sgpf5y6	787	1870	6.24E+02	3.37E+00	0.15	146.43	83.92
pds-20.pre	370	851	1.07E+03	3.38E+00	0.13	51.60	61.08
sgpf5y6.pre	755	1744	1.20E+02	3.22E+00	0.12	295.60	56.57
p19	117	555	4.13E+05	4.37E+04	0.12	156.17	98.19
iiasa	113	262	6.06E+18	6.37E+17	0.12	304.00	46.43
scrs8-2r-256	416	832	1.92E+01	9.13E+00	0.12	214.69	32.78
UMTS	268	828	4.40E+20	3.24E+18	0.11	100.90	103.26
neos818918	265	678	4.29E+02	3.22E+00	0.10	30.12	24.17
rd-rplusc-21	148	454	7.35E+16	4.38E+13	0.10	237.73	239.83
neos4	454	944	2.63E+08	5.59E+06	0.10	176.16	225.01
rosen7	127	649	8.23E+01	5.75E+01	0.10	45.33	155.96
ceria3d	130	647	2.40E+04	1.09E+01	0.10	19.26	32.27
dbir2	157	784	1.71E+06	7.27E+04	0.10	41.07	78.17
scrs8-2r-64	256	512	1.60E+05	1.41E+05	0.10	263.03	68.37
boeing1	122	415	7.36E+10	6.29E+07	0.09	134.16	136.24
scrs8	109	280	2.97E+27	8.93E+24	0.09	233.86	124.79
gesa3_o	148	365	1.73E+29	1.19E+26	0.09	457.90	48.61
neos19	228	487	7.19E+04	5.23E+01	0.09	49.19	126.88
pp08aCUTS	131	332	1.44E+04	4.27E+02	0.08	65.16	41.73
scorpion	131	507	7.19E+06	2.00E+05	0.08	119.88	332.56
gesa3	134	336	1.73E+29	1.19E+26	0.08	169.82	183.58
neos823206	220	547	3.15E+05	2.24E+03	0.08	129.95	97.29
sc205	184	487	1.18E+04	2.10E+01	0.08	105.17	103.36
nsct2	107	544	2.74E+06	2.40E+05	0.07	34.50	48.16
nug05	107	362	1.14E+03	4.15E+00	0.06	5.64	9.54
lpl3	212	461	3.06E+02	2.61E+00	0.06	84.57	80.55

						Relative Run Time	
Matrix Name	n	nnz	Cond	Norm	SPEX Left LU time (ms)	Wiedemann	Lanczos
scrs8-2r-128	192	384	1.92E+01	9.13E+00	0.06	35.22	42.63
scrs8-2c-64	168	336	4.80E+01	2.66E+01	0.06	75.68	94.17
stormg2-8	117	263	4.82E+05	1.77E+03	0.05	96.96	70.89
dcmulti	120	303	7.03E+03	6.26E+02	0.05	92.34	80.35
mkc1	106	250	1.27E+06	2.89E+04	0.05	25.26	32.34
mkc	106	250	1.27E+06	2.89E+04	0.04	26.44	12.39
gesa2_o	102	214	6.47E+09	4.13E+08	0.04	110.27	465.18
scrs8-2r-32	128	256	4.16E+01	3.54E+01	0.04	85.14	90.84
bienst1	102	253	7.61E+02	6.09E+00	0.03	11.11	16.36

Table 6. Comprehensive Results: Tiny Instances