VERIFIED ERROR BOUNDS FOR SPARSE SYSTEMS PART I: THE SPLITTING OF A MATRIX INTO TWO FACTORS[*]

SIEGFRIED M. RUMP[†]

Abstract. Verification methods provide mathematically correct error bounds for the solution of a numerical problem. That includes the proof of solvability of the problem and often uniqueness of the solution within the computed bounds. There are many verification methods for standard problems in numerical analysis, including linear and nonlinear systems of equations, matrix decompositions, eigenproblems, local and global optimization, ordinary and partial differential equations. Many of those verification methods are included in INTLAB, the Matlab/Octave toolbox for reliable computing. Despite several efforts, the solution of general sparse linear systems is an open problem. There are satisfactory algorithms for systems with symmetric positive definite input matrix. To that end error bounds for the solution of $Ax = b$ with general matrix A could be computed using $A^TAx = A^Tb$, but that reduces the applicability in double precision to matrices with condition number up to 10^8.

We give in this note an algorithm for general real or complex sparse matrices with condition number up to the limit 10^{16}. Our algorithm splits into three subalgorithms for symmetric positive definite, symmetric indefinite and general input matrix A. It is based on a mathematically correct lower bound on the smallest singular value $\sigma_{\min}(A)$. A key point is a factorization L_1L_2 such that L_1 and L_2 have identical sets of singular values with the smallest one close to $\sigma_{\min}(A)^{1/2}$. A mathematically correct lower bound on $\sigma_{\min}(L_1) = \sigma_{\min}(L_2)$ is then computed using $L_1^TL_1$. Numerical evidence suggests that bounds for the solution of a linear system are computed for condition numbers up to 10^{16}, and that often the bounds for all entries are close to maximal accuracy, i.e., the bounds differ by few bits.

Based on that an alternative approach will be presented in Part II of this note. Those methods are simpler, but often slower. However, they are sometimes more stable, i.e., may produce verified inclusions where the methods of this Part I fail.

Both approaches for square linear systems will be used in Part II of this note to compute verified error bounds for the solution of least squares problems and for underdetermined linear systems. Inclusions of the solution of general real or complex systems of nonlinear equations with sparse Jacobi matrix are computed by transforming the problem into a linear system with point matrix and interval right hand side.

Key words. sparse linear systems, verification methods, mathematically correct error bounds, lower bound on the smallest singular value, accurate dot products, INTLAB

MSC codes. 65G20, 65F99

1. Introduction.

Standard algorithms to solve numerical problems, e.g. as provided in Matlab [33], are mostly reliable, and usually they produce accurate results. However, there are exceptions. To cite Vel Kahan, *"Numerical problems with standard numerical algorithms are rare; rare enough not to worry about all the time, but not yet rare enough to ignore them"*.

The purpose of verification methods is to provide rigorous error bounds for the solution of numerical problems. The bounds are computed in pure floating-point arithmetic and they are true with mathematical certainty. That includes the proof of solvability of the problem and possibly uniqueness of the solution within the computed bounds.

Verification algorithms are available for many standard numerical problems including systems of linear and nonlinear equations, eigenproblems, local and global optimization, ordinary and partial differential equations, and more. For overviews

[*]Submitted to the editors DATE.

[†]Institute for Reliable Computing, Hamburg University of Technology, Am Schwarzenberg-Campus 3, Hamburg 21073, Germany, and Faculty of Science and Engineering, Waseda University, 3–4–1 Okubo, Shinjuku-ku, Tokyo 169–8555, Japan (rump@tuhh.de).

cf. [37, 49, 41] and the literature cited over there. Many verification algorithms are included in INTLAB [47], the Matlab/Octave toolbox for reliable computing.

For systems of linear equations with full matrix general purpose verification methods are available. They prove to be reliable, i.e., even for ill-conditioned matrices narrow bounds for the solution are computed. For other numerical problems such as ordinary or partial differential equations there is a vast literature, cf. for example [30, 35, 2, 25, 31, 3, 4, 5], however, it seems difficult to provide general purpose verification algorithms.

An open problem, which is part of the *Grand challenges* [38], are verification methods for systems of linear equations with sparse matrix. There are only satisfactory algorithms for systems with symmetric positive definite input matrix.

For given symmetric (positive definite) A it is proposed in [45] to compute an approximation $\tilde{s}$ of the smallest singular value $\sigma_{\min}(A)$ of A, set $s := 0.9\tilde{s}$, factor $B := A - sI$ into $B \approx \tilde{G}\tilde{G}^T$ together with an upper bound e on $\|E\|_1$ for $E := \tilde{G}\tilde{G}^T - B$. Since $\tilde{G}\tilde{G}^T$ is positive semidefinite, it follows that $\|E\|_2 \leqslant \|E\|_1$ because E is symmetric and

$$\sigma_{\min}(A) = \sigma_{\min}(\tilde{G}\tilde{G}^T + sI - E) \geqslant \sigma_{\min}(\tilde{G}\tilde{G}^T + sI) - \|E\|_2 \geqslant s - e\ . \tag{1.1}$$

We put "positive definiteness" in quotes because it is not a prerequisite for the method but follows a posteriori. Later (cf. [53]) that method used a priori estimates on $\|E\|_2$ based on Demmel's result [9], see also [14, Theorem 10.5]. If $\sigma_{\min}(A) \geqslant \alpha > 0$, then A is nonsingular, and for an approximate solution $\tilde{x}$ of a linear system $Ax = b$ it follows

$$\|A^{-1}b - \tilde{x}\|_\infty \leqslant \|A^{-1}b - \tilde{x}\|_2 \leqslant \alpha^{-1}\|b - A\tilde{x}\|_2.$$

The method in (1.1) might be applied to A^TA for general A, however, that squares the condition number and limits applications to $\operatorname{cond}(A) \lesssim 10^8$ in double precision (binary64). That is the reason why [49, Challenge 10.15] asks for a verification method for sparse linear systems of reasonable size with $\operatorname{cond}(A) \geqslant 10^{10}$.

Most methods to solve full linear systems use an approximate inverse as preconditioner which is prohibitive for sparse system matrix. The method [40] replaces an approximate inverse by the approximate solution of n linear systems with the columns of the identity matrix as right hand side.

For general symmetric sparse matrix a factorization $A \approx \tilde{L}_1\tilde{L}_2^T$ obtained by factoring $D = D_1D_2$ of an LDL^T factorization and setting $\tilde{L}_1 := LD_1$ and $\tilde{L}_2 := LD_2^T$ was proposed in [45], and similarly $A \approx \tilde{L}\tilde{M}^T$ for general A with computing $\tilde{L}$ and $\tilde{M}$ by an LU-decomposition. Lower bounds of $\sigma_{\min}(A)$ follow by

$$\sigma_{\min}(A) \geqslant \sigma_{\min}(\tilde{L}_1)\sigma_{\min}(\tilde{L}_2) - \|A - \tilde{L}_1\tilde{L}_2^T\|_2$$

and similarly for $A \approx \tilde{L}\tilde{M}^T$, where the lower bounds on the smallest singular value of the factors follow by applying (1.1) to $\tilde{L}_1^T\tilde{L}_1 - \tilde{s}I$ and so forth. If the condition numbers of a factor F is of the order $\operatorname{cond}(A)^{1/2}$, then $\operatorname{cond}(F^TF) \approx \operatorname{cond}(A)$ and those methods work fine. However, not too many details were given in [45].

Next we proved the following theorem [46, Theorem 1.1]:

Theorem 1.1. *Let symmetric $A \in \mathbb{R}^{n\times n}$, $0 < \tilde{\lambda} \in \mathbb{R}$ and $\tilde{L}_1, \tilde{D}_1, \tilde{L}_2, \tilde{D}_2 \in \mathbb{R}^{n\times n}$ be given. If the inertia of $\tilde{D}_1$ and $\tilde{D}_2$ are equal, then for any matrix norm*

$$\sigma_{\min}(A) > \tilde{\lambda} - \max\{\|A - \tilde{\lambda}I - \tilde{L}_1\tilde{D}_1\tilde{L}_1^T\|, \|A + \tilde{\lambda}I - \tilde{L}_2\tilde{D}_2\tilde{L}_2^T\|\}. \tag{1.2}$$

If all eigenvalues of $\tilde{D}_1$ are positive, then

$$\sigma_{\min}(A) > \tilde{\lambda} - \|A - \tilde{\lambda}I - \tilde{L}_1\tilde{D}_1\tilde{L}_1^T\|. \tag{1.3}$$

This approach needs two LDL^T-decompositions and is applicable for condition numbers of A close to $\mathbf{u}^{-1} \approx 10^{16}$. In [48] it was proposed to apply Theorem 1.1 to the augmented matrix $B := \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$. That symmetric matrix has the same condition number as A because its eigenvalues are $\pm\sigma_i(A)$. For the time being the approaches in [45, 46, 48] were not further pursued because the symmetric pivoting of the LDL^T-decomposition was not stable enough.

Nowadays good scaling and equilibration routines are available [11, 12] making those methods attractive. That was observed by Terao and Ozaki [57] and triggered our note in two parts. They proposed to apply the idea in Theorem 1.1 to the augmented matrix B. For an approximation $\tilde{s}$ of the smallest singular value of B they compute $\tilde{L}\tilde{D}\tilde{L}^T \approx B - sI$ with $s := 0.5\tilde{s}$. Since for nonsingular A the inertia of B is known to be $(-n, 0, n)$, the lower bound on $\sigma_{\min}(A) = \sigma_{\min}(B) \geqslant \tilde{s} - \|B - \tilde{L}\tilde{D}\tilde{L}^T\|_2$ follows if the inertia of $\tilde{D}$ is $(-n, 0, n)$ as well. They use in particular the preconditioning in [11] to ensure stability of the LDL^T-decomposition. However, only the factors $\tilde{L}, \tilde{D}$ of the shifted matrix $B - \tilde{s}I$ are available, not of B itself. It was proposed and analysed in [53] that nevertheless a residual iteration based on $\tilde{L}, \tilde{D}$ works, and that is used by Terao and Ozaki [57].

In this note we treat three cases separately, namely symmetric (positive definite), symmetric indefinite and general matrices. For the first case we improve the bound (1.1) in [53] utilizing sparsity and Perron-Frobenius Theory. For the second case we factor a symmetric matrix A into $A \approx F_1F_2$ with F_1, F_2 having identical sets of singular values, and numerical evidence suggesting $\text{cond}(F_1) \approx \text{cond}(A)^{1/2}$. Then we apply (1.1) to $F_1F_1^T$ to compute a lower bound α on $\sigma_{\min}(F_1) = \sigma_{\min}(F_2)$, such that $\sigma_{\min}(A) \geqslant \alpha^2 - \|A - F_1F_2\|_2$. For general matrices we use a similar scheme for the augmented matrix B.

In all three cases the matrix A (or the augmented matrix B) is expressed as the product of two matrices F_1F_2. In contrast to $A = LDL^T$ this bears the advantage that the entries of the residual $A - F_1F_2$ (or $B - F_1F_2$) are one dot product each. Thus an inclusion of good quality can be computed using one of the many accurate dot product algorithms [32, 36, 10, 39, 61, 60]. In contrast, an inclusion of $A - LDL^T$ is computed in two steps with an interval factor in the second product.

We want to stress that there is hardly a general purpose algorithm to solve sparse linear systems. Indeed we tried many examples from the Suite Sparse Matrix Collection [8] and found linear systems where our verification method is by two orders of magnitude faster than the built-in backslash Matlab operator (but also vice versa). That should not happen because our verifications methods include an approximate solution of the linear system.

As test matrices we took all real square matrices of the Suite Sparse Matrix Collection with dimension n satisfying $10^4 \leqslant n \leqslant 10^6$ and estimated condition number κ with $10^{10} \leqslant \kappa \leqslant 10^{16}$. That resulted in 306 test cases. In 300 cases we could compute accurate verified inclusions of the solution, usually about a factor 3 to 10 slower than Matlab's backslash operator, but also sometimes faster. That is the price we pay for mathematically rigorous bounds.

Our primary target is that our algorithm ends successfully, i.e., verifies non-singularity of the input matrix and computes error bounds for the solution of the linear system. Our algorithm is tuned to that goal accepting some penalty in computing time. Besides the mathematically rigorous verification, the second focus is to compute accurate bounds for the solution, in most cases with maximum relative error $\lesssim 10^{-15}$, i.e., close to maximally accurate bounds in double precision (bainry64). That allowed to compute the relative error of the approximation produced by Matlab's backslash operator. That was often of the order 10^{-8}, but also worse. In many cases our algorithm was twice as fast and more accurate than the method proposed in [57].

We assume a set of floating-point numbers $\mathbb{F}$ with an arithmetic according to the IEEE754 floating-point standard [18] to be given. We use double precision (binary64) in a nearest rounding[1] with relative rounding error unit $\mathbf{u} = 2^{-53} \approx 10^{-16}$, and we use directed rounding downwards (towards $-\infty$) and upwards (towards $+\infty$). We use float$(\cdot)$ to indicate the result of an expression with all operations executed in floating-point. If the order of execution is not unique, results are true for any order. The error of a single operation $\circ \in \{+,-,\times,/\}$ of floating-point numbers a, b is bounded by [14]

$$|\text{float}(a \circ b) - a \circ b| \leqslant \mathbf{u} \cdot \min\left(\ |a \circ b| \ , \ |\text{float}(a \circ b)| \ \right) . \tag{1.4}$$

For $\circ \in \{+,-\}$ this is also true for compatible vectors or matrices a, b with comparison and absolute value to be understood entrywise. When using a directed rounding (1.4) remains true when replacing $\mathbf{u}$ by $2\mathbf{u}$.

Our goal is to calculate mathematically correct but also accurate inclusions for the solution of a sparse linear system $Ax = b$. To that end we use the following notations:

$$\begin{array}{ll} [\![expr]\!]_{2,1} & \text{evaluation in extended precision, result rounded into } \mathbb{F} \\ \langle expr \rangle & \text{inclusion computed using directed roudings in } \mathbb{F} \\ \langle\!\langle expr \rangle\!\rangle_{2,1} & \text{inclusion computed in extended precision and rounded into } \mathbb{F} \end{array} \tag{1.5}$$

We added the subscripts $_{2,1}$ to emphasize that the evaluation is performed in extended precision but the result is rounded into working precision, i.e., into $\mathbb{F}$.

The notations in (1.5) are used exclusively for expressions where each entry is computable by a dot product. For the two latter notations for inclusions the expression has to satisfy an additional property: When computing the expression in rounding downwards, then the computed result is a mathematically correct lower bound of the true result, and similarly for rounding upwards. Typical examples for $[\![\cdot]\!]_{2,1}$ are $Ax - b$ or $A - R^T R$. The second expression is not suitable for $\langle\cdot\rangle$ or $\langle\!\langle\cdot\rangle\!\rangle_{2,1}$ because the result computed in rounding downwards is not necessarily a correct lower bound of the true result. It becomes suitable by rewriting it into $R^T R - A$.

For the implementation of $[\![\cdot]\!]_{2,1}$ and $\langle\!\langle\cdot\rangle\!\rangle_{2,1}$ any of the many accurate dot product algorithms is suitable. There is a new, very fast Matlab implementation which will be used in Part II of this note.

In [57] the toolbox Advanpix [15] was used, a multiple-precision Matlab package emulating a large number of Matlab's algorithms. In order to have a fair comparison with [57] we used [15] in this note as well. The number `d` of decimal digits of precision can be freely specified by `mp.Digits(d)`. The package includes a particularly fast implementation of extended precision arithmetic to be specified by `mp.Digits(34)` with relative rounding error unit 2^{-113}. This precision is what we are using throughout this note. Sample executable Matlab/INTLAB codes for the expressions in (1.5) for

[1] Our results in rounding to nearest are true for any rounding of ties.

$Ax - b$ are

(1.6)
```
⟦expr⟧_{2,1}   res = double(A * mp(x) - b);
⟨expr⟩         setround(-1); resinf = A * x - b;
               setround(+1); ressup = A * x - b;
               res = infsup(resinf, ressup);
⟪expr⟫_{2,1}   setround(-1); resinf = double(A * mp(x) - b);
               setround(+1); ressup = double(A * mp(x) - b);
               res = infsup(resinf, ressup);
```

Note that the type cast `mp(x)` ensures that `A*mp(x)` is computed in extended precision with extended precision result, and in turn that ensures that the difference in `A*mp(x)-b` is computed in extended precision as well. Moreover, the typecast `double(·)` in the implementation of $\langle\!\langle\cdot\rangle\!\rangle_{2,1}$ respects the rounding mode so that `resinf`$\leqslant Ax - b \leqslant$`ressup` holds true.

It is common to use $\|P\|_2 \leqslant \sqrt{\|P\|_1\|P\|_\infty}$ to bound the spectral norm of a matrix P. However, Perron-Frobenius Theory and [7] imply for any positive vector x the better bound

$$\|P\|_2 \leqslant \||P|\|_2 = \sigma_{\max}(|P|) = \sqrt{\lambda_{\max}(|P|^T|P|)} \leqslant \max_k \frac{\big(|P|^T(|P|x)\big)_k}{x_k} \tag{1.7}$$

for general P and

$$\|P\|_2 \leqslant \max_k \frac{(|P|x)_k}{x_k} \tag{1.8}$$

for symmetric/Hermitan P. To that end we used in [52] the following algorithm:

(1.9)
```
function N = NormBnd(A, herm)
  x = ones(size(A, 1), 1); M = [12]; iter = 0; A = mag(A);
  while(abs(diff(M)/sum(M)) > .1) && (iter < 10)
    iter = iter + 1;
    y = A * x;
    if herm, y = A' * y; end
    x = y./x;
    M = [min(x)max(x)];
    scale = max(y);
    x = max(y/scale, 1e - 12);
  end
  setround(1)
  if herm, N = max((A * x)./x); else N = max(sqrt((A' * (A * x))./x)); end
end
```

That algorithm is used in [57] as well. Compared to `sqrt(norm(A,1)*norm(A,inf))` numerical evidence suggests that few power iterations in (1.9) starting with the vector x of all $1's$ is faster and improves the bound by a factor 2.

We use standard eigenvalue perturbation bounds [58] for symmetric or Hermitian $n \times n$ matrices A, E, i.e.,

$$\lambda_k(A) + \lambda_n(E) \leqslant \lambda_k(A+E) \leqslant \lambda_k(A) + \lambda_1(E) \quad \Rightarrow \quad |\lambda_k(A+E) - \lambda_k(A)| \leqslant \|E\|_2 \tag{1.10}$$

for $\lambda_1 \geqslant \ldots \geqslant \lambda_n$ denoting the eigenvalues and $k \in \{1, \ldots, n\}$. Moreover, for $A, B \in \mathbb{R}^{n\times n}$ we use [17, Theorem 3.3.16]

$$\sigma_{\min}(AB) \geqslant \sigma_{\min}(A)\sigma_{\min}(B) \ . \tag{1.11}$$

A real or complex signature matrix S is diagonal with $|S_{kk}| = 1$ for all k. For vectors (and similarly for matrices) we use $|\cdot|$ for the vector of absolute values, and $x \leqslant y$ denotes entrywise comparison.

We begin this note with some improved floating-point error estimates on matrix products, on the 2-norm of residuals and an a priori error estimate of Cholesky decomposition, improving on the mostly used $\gamma_k := \frac{k\mathbf{u}}{1-k\mathbf{u}}$, cf. [14]. In particular we present computable bounds on the error of matrix products and residuals when using directed rounding. In the following sections we introduce our methods for linear systems with symmetric (positive definite), with symmetric indefinite, and with general matrix. All three methods are based on the computation of a lower bound of the smallest singular value of some symmetric (Hermitian) matrix. We discuss how to obtain an approximation of the smallest singular value, and we show how a true lower bound is used to obtain rigorous and sharp error bounds for $A^{-1}b$.

Extra sections discuss scaling and equilibration, as well as some factorization of Hermitian 2×2 matrices. We show how to handle complex linear systems, data afflicted with tolerances, and present Algorithm `VerifySparselss` to compute rigorous error bounds for a linear system with real or complex sparse matrix and multiple right hand sides. This is our main algorithm and it chooses between subalgorithms for symmetric (positive definite), symmetric indefinite and general matrix, and real or complex data. We compare our algorithm with that in [57] and close the paper with a compilation of computational results.

2. Floating-point error estimates.

The result c of a floating-point operation is called faithful if there is no other floating-point number between c and the true real result. In IEEE754 operations with rounding to nearest, towards $\pm\infty$ or towards zero are faithful. We begin with error bounds for the computed approximation of dot products and matrix products.

For $x, y \in \mathbb{F}^n$ with at most μ nonzero products the linear estimate

$$|\text{float}(x^T y) - x^T y| \leqslant \mu\mathbf{u}|x|^T|y| \tag{2.1}$$

was shown in [23]. The bound is true for any order of evaluation of $x^T y$ and without restriction on the dimension n. Hence, the error of the floating-point approximation of AB for $A \in \mathbb{F}^{m\times k}, B \in \mathbb{F}^{k\times n}$ is bounded by

$$|\text{float}(AB)_{ij} - (AB)_{ij}| \leqslant \mu\mathbf{u}(|A||B|)_{ij} \tag{2.2}$$

for μ denoting the maximum number of nonzero products to compute the entries of AB. To obtain a computable bound using (2.2) the extra matrix product $P := |A||B|$ with error bound is necessary. That extra matrix product can be avoided by using directed rounding. To that end we need an error estimate like (2.2) for floating-point dot products with directed rounding. In that case a restriction of k is mandatory because in rounding upwards, for example, the result of $1+e$ for tiny positive e is the successor of 1, so that the error is about $2\mathbf{u}$.

The first bound for directed rounding was given by Ozaki [42], namely $|\mathrm{float}(AB)-AB| \leqslant 2(\mu+4)\mathbf{u}AB$. It was designed for mixed-precision calculations. The bound requires $4\mu \leqslant \mathbf{u}$ but also that both A, B are nonnegative. For general A, B it was shown in [27, Corollary 4] that

$$|\mathrm{float}(AB)_{ij} - (AB)_{ij}| \leqslant 2\mu\mathbf{u}(|A||B|)_{ij} \tag{2.3}$$

is true for computing float(AB) using a faithful rounding provided that $\mu \leqslant (2\mathbf{u})^{-1/2}$.

The assumption $\mu \leqslant (2\mathbf{u})^{-1/2}$ bounding the number of nonzero products seems hardly an obstacle when using double precision (binary64), i.e. $\mu \leqslant 2^{26} = 67,108,864$ nonzero products per entry. But if so, the following Lemma 2.1 may be used up to $\mu \leqslant 2,251,799,813,685,248 \approx 2.2\cdot 10^{15}$ nonzero products per entry. Note that it is mandatory to bound the number of nonzero products μ, cf. [27].

Lemma 2.1. *Let $A \in \mathbb{F}^{m\times k}$ and $B \in \mathbb{F}^{k\times n}$ be given, and let float(AB) be calculated in a faithful-rounding. Denote by μ the maximum number of nonzero products to compute the entries of AB. If $2(\mu-1)\mathbf{u} \leqslant 1$, then*

$$|\mathit{float}(AB)_{ij} - (AB)_{ij}| \leqslant (2\mu+1)\mathbf{u}(|A||B|)_{ij}\ . \tag{2.4}$$

Proof. Let $z \in \mathbb{F}^n$ be a vector of floating-point numbers, and let float$(\sum_{k=1}^n z_k)$ be computed in some faithful rounding in any order. Then [26, Corollary 3.3] shows

$$|\mathrm{float}(\sum_{k=1}^n z_k) - \sum_{k=1}^n z_k| \leqslant 2(\mu-1)\mathbf{u}\sum_{k=1}^n |z_k| \tag{2.5}$$

provided that the vector z has not more than μ nonzero elements. Let $x, y \in \mathbb{F}^n$ be given, denote $z_k := \mathrm{float}(x_k y_k)$ for $k \in \{1,\ldots,n\}$, and let $\mathrm{float}(x^T y) = \mathrm{float}(\sum_{k=1}^n z_k)$, all computed in some faithful rounding. Then

$$|\mathrm{float}(x_k y_k) - x_k y_k| \leqslant 2\mathbf{u}|x_k y_k| \quad \text{and} \quad |z_k| = |\mathrm{float}(x_k y_k)| \leqslant (1+2\mathbf{u})|x_k y_k|\ .$$

Hence the definition of μ and using $2(\mu-1)\mathbf{u} \leqslant 1$ shows

$$\begin{aligned} |\mathrm{float}(x^T y) - x^T y| &\leqslant |\mathrm{float}(\textstyle\sum_{k=1}^n z_k) - \sum_{k=1}^n z_k| + |\sum_{k=1}^n (z_k - x_k y_k)| \\ &\leqslant 2(\mu-1)\mathbf{u}\textstyle\sum_{k=1}^n |z_k| + 2\mathbf{u}\sum_{k=1}^n |x_k y_k| \\ &\leqslant [2(\mu-1)\mathbf{u}(1+2\mathbf{u}) + 2\mathbf{u}]\,|x^T||y| \\ &\leqslant 2(\mu+1)\mathbf{u}|x^T||y| \end{aligned}$$

and the result follows by applying this estimate to each entry of AB. ∎

We start with a mathematically correct a priori error bound for a matrix product AB and for a residual $AB - C$ without computing $|A||B|$.

Lemma 2.2. *Let $A \in \mathbb{F}^{m\times \ell}$ and $B \in \mathbb{F}^{\ell\times n}$ be given, and let μ_i and ν_j denote the number of nonzero elements in the i-th row of A and the j-th column of B, respectively. Furthermore, denote by ϱ_i and σ_j the Euclidean norm of the i-th row of A and the j-th column of B, respectively. Then using a nearest-rounding and any order of evaluation*

$$\|\mathit{float}(AB) - AB\|_2 \leqslant \mathbf{u}\sum_{k=1}^n \min(\mu_k,\nu_k)\rho_k\sigma_k \tag{2.6}$$

without limit on n. For $C \in \mathbb{F}^{m\times n}$ and $E := \mathit{float}(AB - C)$ it follows

$$(2.7) \qquad \|\mathit{float}(AB - C) - (AB - C)\|_2 \leqslant \mathbf{u}\left(\|E\|_2 + \sum_{k=1}^{n} \min(\mu_k, \nu_k)\rho_k\sigma_k\right)$$

without limit on n. Denote by μ the maximum number of nonzero products in the products $(AB)_{ij}$. If a faithful-rounding is used and $\mu \leqslant (2\mathbf{u})^{-1/2}$, then (2.6) and (2.7) remain true when replacing $\mathbf{u}$ by $2\mathbf{u}$. For faithful-rounding and $2(\mu-1)\mathbf{u} \leqslant 1$, (2.6) and (2.7) remain true when replacing $\mathbf{u}$ by $2\mathbf{u}$ and $\min(\mu_k, \nu_k)$ by $\min(\mu_k, \nu_k) + 1$.

Proof. The computation of the element $(AB)_{ij}$ involves at most $\min(\mu_i, \nu_j)$ nonzero products. Hence (2.2) implies for a nearest-rounding

$$|\mathrm{float}(AB)_{ij} - (AB)_{ij}| \leqslant \min(\mu_i, \nu_j)\mathbf{u}(|A||B|)_{ij} \leqslant \min(\mu_i, \nu_j)\mathbf{u}\varrho_i\sigma_j \ .$$

Let $\widehat{\varrho}$ and σ denote the column vectors with elements $\mu_i\varrho_i$ and σ_j, respectively. Then using the outer product $\widehat{\rho}\sigma^T$ it follows

$$\|\mathrm{float}(AB) - AB\|_2 \leqslant \|\,|\mathrm{float}(AB) - AB|\,\|_2 \leqslant \|\widehat{\varrho}\sigma^T\|_2\mathbf{u} = \sigma^T\widehat{\rho}\mathbf{u} = \mathbf{u}\sum_{k=1}^{n}\sigma_k\mu_k\rho_k.$$

Denoting similarly by $\widehat{\sigma}$ the column vector with elements $\nu_j\sigma_j$ gives

$$\|\mathrm{float}(AB) - AB\|_2 \leqslant \widehat{\sigma}^T\rho\mathbf{u} = \mathbf{u}\sum_{k=1}^{n}\nu_k\sigma_k\rho_k$$

and implies (2.6). Using $P := \mathrm{float}(AB)$ and (1.4) gives

$$\begin{array}{rcl} \|\mathrm{float}(AB - C) - (AB - C)\|_2 & = & \|\mathrm{float}(P - C) - (AB - C)\|_2 \\ & = & \|\mathrm{float}(P - C) - (P - C) + (P - AB)\|_2 \\ & \leqslant & \mathbf{u}\|E\|_2 + \|P - AB)\|_2 \end{array}$$

and proves (2.7). For faithful rounding the estimates follow by (2.3) and (2.4). $\square$

The application of Lemma 2.2 is as follows. We compute `M1 = A*B` in rounding upwards with the estimate $\alpha := 2\mathbf{u}\sum_{k=1}^{n}\min(\mu_k, \nu_k)\rho_k\sigma_k$ as in (2.3). That is an a priori bound for the error of $\|\mathrm{float}(AB) - AB\|$. If not sufficiently accurate, we calculate `M0 = A*B` in rounding downwards. Hence $M_0 \leqslant AB \leqslant M_1$ implies the improved a posteriori bound $\|\mathrm{float}(AB) - AB\|_2 \leqslant \|\max(|M_0|, |M_1|)\|_2$.

Corollary 2.3. *Let $A \in \mathbb{F}^{n\times n}$ be given and denote by μ_k the number of nonzero elements in the k-th row of A. Then for a nearest-rounding*

$$(2.8) \qquad \|\mathit{float}(AA^T) - AA^T\|_2 \leqslant \mathbf{u}\sum_{k=1}^{n}\mu_k(AA^T)_{kk}$$

is true without limit on n. If $\max\mu_k \leqslant (2\mathbf{u})^{-1/2}$ and rounding upwards is used, then

$$(2.9) \qquad \|\mathit{float}(AA^T) - AA^T\|_2 \leqslant \mathbf{u}\sum_{k=1}^{n}\mu_k\left(\mathit{float}(AA^T)\right)_{kk}.$$

If $\max\mu_k \leqslant \mathbf{u}^{-1}/2$, then (2.9) remains true when replacing μ_k by $\mu_k + 1$.

Proof. Denote by ϱ_k the Euclidean norm of the k-th row of A. Then Lemma 2.2 implies

$$\|\text{float}(AA^T) - AA^T\|_2 \leqslant \mathbf{u}\sum_{k=1}^{n}\mu_k\rho_k^2 = \mathbf{u}\sum_{k=1}^{n}\mu_k(AA^T)_{kk}\ .$$

In rounding upwards $(AA^T)_{kk} \leqslant \big(\text{float}(AA^T)\big)_{kk}$ and the results follows. ◻

We often need estimates of a residual. For example, if $C \approx AB$ is a decomposition, we need an upper bound for $\|C - AB\|_2$. We compute that bound in three stages. First, we use the a priori estimate in (2.7). If not successful, then we compute a better bound using an inclusion of $C - AB$ obtained by using rounding downwards and upwards. If still not successful, accurate dot products are used.

Next we list executable Matlab code for the three stages to compute upper bounds for the spectral norm of a general residual $C-AB$. That is sufficient for our verification methods because we construct decompositions with two factors by transforming, e.g., $M \approx LDL^T$ into $M \approx L_1L_2$. We assume that the maximum number μ_k of nonzero products in the computation of the entries of AB is restricted by $\max\mu_k \leqslant (2\mathbf{u})^{-1/2} = 67,108,864$. If only $\max\mu_k \leqslant \mathbf{u}^{-1}/2 \approx 4.5\cdot 10^{15}$ is satisfied, then the code is adapted following Corollary 2.3.

LEMMA 2.4. *Let $A \in \mathbb{F}^{m\times k}$, $B \in \mathbb{F}^{k\times n}$ and $C \in \mathbb{F}^{m\times n}$. Then executing the Matlab code*

(2.10)
```
setround(1); Q = A * B - C;
mu = sum(spones(A), 2); nu = sum(spones(B));
rho = vecnorm(A, 2, 2); sigma = vecnorm(B, 2);
errAB = (min(mu', nu). * sigma) * rho;
alpha = NormBnd(Q, false) + pow2(-52) * (NormBnd(C, false) + errAB);
```

implies $\|C - AB\|_2 \leqslant \alpha$. Executing after (2.10) *the Matlab code*

(2.11)
```
setround(-1); Q = max(Q, abs(A * B - C));
beta = NormBnd(Q, false);
```

implies $\|C - AB\|_2 \leqslant \beta$. Furthermore, after executing

(2.12)
```
setround(0); mp.Digits(34);
F = C - mp(A) * B; u = pow2(-53); v = pow2(-113);
setround(1); G = double(abs(F));
mu = sum(spones(A), 2); nu = sum(spones(B));
rho = vecnorm(A, 2, 2); sigma = vecnorm(B, 2);
normG2 = NormBnd(G, false);
errAB = (min(mu', nu). * sigma) * rho;
gamma = normG2  +  v * (normG2 + errAB);
```

it follows $\|C - AB\|_2 \leqslant \gamma$. Finally, let $A \in \mathbb{F}^{n\times k}$, $B = SA^T$ for a signature matrix

$S \in \mathbb{F}^{k\times k}$ and $C \in \mathbb{F}^{n\times n}$. Then executing

(2.13)

```
setround(0); mp.Digits(34);
F = C − mp(A) ∗ B; v = pow2(−113);
setround(1); G = double(abs(F));
normG2 = NormBnd(G, false);
errAtA = sum(spones(A), 2)′ ∗ sqr(vecnorm(A, 2, 2));
alpha = normG2 + v ∗ (normG2 + errAtA);
```

implies $\|C - AB\|_2 \leqslant \alpha$.

Remark 2.5. In order to compute mathematically correct bounds directed roundings are used. Moreover, in the calls of `NormBnd` from (1.9) the second parameter can be replaced by `true` for Hermitian input. In a practical implementation the three occurrences of the matrix `G` in (2.12) would be replaced by one matrix `F` to save memory, in particular for large and sparse input A, B, C.

Remark 2.6. For the codes in (2.12) it is not necessary to compute upper bounds for the Euclidean norms ϱ_i and σ_j in extended precision because these computations are perfectly well conditioned. Note that the computation of μ and ν is error-free.

Proof. For the first code (2.10) the rounding upwards implies that the computed quantities `mu, nu, rho, sigma` are upper bounds of $\mu, \nu, \varrho, \sigma$ in Lemma 2.2, so that (2.7) proves $\|C - AB\|_2 \leqslant \alpha$. Note that $2\mathbf{u} = 2^{-52}$ is used because of upward directed rounding. For (2.11) let

```
setround(−1); Q1 = A ∗ B − C;
setround(+1); Q2 = A ∗ B − C;
```

Note that `Q2` is the matrix `Q` in (2.10) and `Q1` is implicitly computed in (2.11). Then the rounding modes imply[2] $\texttt{Q1} \leqslant AB - C \leqslant \texttt{Q2}$ with entrywise comparison. Hence $|AB - C| \leqslant \max(|\texttt{Q1}|, |\texttt{Q2}|)$ and $\|C - AB\|_2 \leqslant \beta$ follows.

The third code (2.12) uses the multiple precision toolbox [15] and computes the residual `F = C - mp(A)*B` in extended precision and rounding to nearest with relative rounding error $\mathbf{v} := 2^{-113}$. The rounding upwards in the third line implies that the quantities `mu, nu, rho, sigma` are upper bounds of $\mu, \nu, \varrho, \sigma$ in Lemma 2.2. Denote $M := \texttt{mp(A)} * \texttt{B}$. Then $F = \mathrm{fl}(C - M)$ and (2.6) implies

$$(2.14) \qquad \|C - AB\|_2 \leqslant \|C - M + M - AB\|_2 \leqslant (1 + \mathbf{v})\|F\|_2 + \mathbf{v}\sum_{k=1}^{n} \min(\mu_k, \nu_k)\rho_k\sigma_k \ .$$

The toolbox Advanpix [15] respects the rounding mode, in particular the type cast `double` from `mp`-tpye to binary64. Hence the double precision matrix G satisfies $|F| \leqslant G$ by the third line, and therefore $\|F\|_2 \leqslant \||F|\|_2 \leqslant \|G\|_2 \leqslant \texttt{normG2}$ and $\|C - AB\|_2 \leqslant \gamma$.

The fourth code (2.13) uses again the multiple precision toolbox [15]. By assumption the set of nonzero elements of A and B are identical, and rows and corresponding columns of A and B have the same Euclidean length. When using the code (2.12) to bound $\|C - AB\|_2$, then

```
mu = sum(spones(A), 2) = sum(spones(B)) = nu′
```

[2]Note that this is true for using `A*B-C`, but would not necessarily be true when using `C-A*B`.

and

$$\texttt{rho} = \texttt{vecnorm(A,2,2)} = \texttt{vecnorm(B,2)}' = \texttt{sigma}'$$

and the result follows. ∎

Note that (2.14) implies that $\|C-AB\|_2$ is very close to $\|F\|_2$ and therefore to $\|G\|_2$, so that the overestimation of the computed γ in (2.12) is basically $\|G\|_2 \leqslant \max_k \sigma_k(|G|)$.

For the special case of Cholesky decomposition $A \approx \tilde{R}^T\tilde{R}$ there is an a priori estimate [53, Lemma 2.2], [14, Theorem 10.5] of the residual $\|\tilde{R}^T\tilde{R} - A\|_2$ without computing $\tilde{R}^T\tilde{R}$. We improve this estimate by applying Perron-Frobenius Theory.

Lemma 2.7. *Let symmetric $A \in \mathbb{F}^{n\times n}$ be given and assume that the floating-point Cholesky factorization of A runs to completion. Denote the computed factor by $\tilde{R}$, and let the vector $\mu \in \mathbb{N}^n$ consist of μ_i denoting the number of nonzero elements in the i-th column of $\tilde{R}$ and assume $\mathbf{u}\max\mu_k < 1$. Denote by $\Phi \in \mathbb{R}^{n\times n}$ the matrix with $\Phi_{ij} := \min(\mu_i,\mu_j)+1$ and by $D \in \mathbb{R}^{n\times n}$ the diagonal matrix with $D_{kk} = \left(\frac{A_{kk}}{1-\Phi_{kk}\mathbf{u}}\right)^{1/2}$. Then for a nearest-rounding in the absence of underflow and overflow $\Delta A := \tilde{R}^T\tilde{R} - A$ satisfies*

$$\|\Delta A\|_2 \leqslant \mathbf{u}\|D\Phi D\|_2. \tag{2.15}$$

If a faithful-rounding is used and $\max\mu_k \leqslant (2\mathbf{u})^{-1/2}$, then the estimate remains true when replacing $\mathbf{u}$ by $2\mathbf{u}$.

Remark 2.8. The matrix Φ is a full matrix. Hence computing (2.15) seems to be costly, in particular for sparse A. However, Φ has a special structure which is utilized in Corollary 2.9 to compute an improved upper bound for $\|\Delta A\|_2$ efficiently.

Proof. In [51] it was shown that

$$|\Delta A|_{ij} \leqslant (i+1)\mathbf{u}(|\tilde{R}^T||\tilde{R}|)_{ij}$$

for $1 \leqslant i,j \leqslant n$. The number of nonzero products in the computation of $\tilde{R}_{ij}$ does not exceed $\min(\mu_i,\mu_j)$, plus a square root in case $i=j$. Using the improved error estimate in Lemma 2.2 and carefully going through the proof of Theorem 4.4 in [51] gives

$$|\Delta A|_{ij} \leqslant \varphi_{ij}\mathbf{u}(|\tilde{R}^T||\tilde{R}|)_{ij} \quad \text{for } \varphi_{ij} := \min(\mu_i,\mu_j)+1.$$

Following the proof of [14, Theorem 10.5] denote the i-th column of $\tilde{R}$ by $\tilde{r}_i$. Then

$$\|\tilde{r}_i\|_2^2 = \tilde{r}_i^T\tilde{r}_i \leqslant A_{ii} + |\Delta A_{ii}| \leqslant A_{ii} + \varphi_{ii}\mathbf{u}\tilde{r}_i^T\tilde{r}_i$$

and $\|\tilde{r}_i\|_2^2 \leqslant (1-\varphi_{ii}\mathbf{u})^{-1}A_{ii}$. Then Cauchy-Schwarz's inequality implies

$$\begin{aligned} |\Delta A|_{ij} &\leqslant \varphi_{ij}\mathbf{u}|\tilde{r}_i^T||\tilde{r}_j| \leqslant \varphi_{ij}\mathbf{u}\|\tilde{r}_i\|_2\|\tilde{r}_j\|_2 \\ &\leqslant \left(\frac{A_{ii}}{1-\varphi_{ii}\mathbf{u}}\right)^{1/2}\varphi_{ij}\left(\frac{A_{jj}}{1-\varphi_{jj}\mathbf{u}}\right)^{1/2}\mathbf{u} \leqslant (D\Phi D)_{ij}\,\mathbf{u} \end{aligned} \tag{2.16}$$

and proves (2.15) and the lemma. ∎

By definition $D\Phi D$ is symmetric positive definite, so $\|D\Phi D\|_2$ is equal to the largest eigenvalue, i.e., the Perron root of $D\Phi D$. Hence $D\Phi D \geqslant 0$ and Perron-Frobenius Theory [7], [16, Theorem 8.1.26] imply

$$\|D\Phi D\|_2 \leqslant \max_k \frac{(D\Phi Dx)_k}{x_k} \qquad \text{for every positive } x \in \mathbb{R}^n\ . \tag{2.17}$$

Moreover, a power iteration converges monotonically to $\|D\Phi D\|_2$ for any positive starting vector x. A problem is, however, that the matrix Φ if full. Fortunately, the product Φx for $x \in \mathbb{F}^n$ can be computed efficiently as follows. My dearest thanks to Marko Lange [28] who provided the ingenious piece of Matlab code in (2.18).

COROLLARY 2.9. *Let $0 < v \in \mathbb{R}^n$ be sorted in ascending order and define $\Phi \in \mathbb{R}^{n\times n}$ by $\Phi_{ij} := \min(v_i, v_j)$. Then for $x \in \mathbb{R}^n$ the vector* `w` *computed by the code*

$$\begin{aligned} &\texttt{rcx} = \texttt{cumsum}(\texttt{x}, 1, '\texttt{reverse}');\\ &\texttt{vx} = \texttt{v.} * \texttt{x};\\ &\texttt{w} = \texttt{cumsum}(\texttt{vx}) - \texttt{vx} + \texttt{v.} * \texttt{rcx}; \end{aligned} \tag{2.18}$$

is equal to Φx.

It is not difficult to verify that indeed `w` $= \Phi x$. The requirement that v is sorted is crucial, and that is no obstacle because of the definition of Φ.

The previous estimate [53, Lemma 2.2], [14, Theorem 10.5] continues from (2.16) by replacing the entries φ_{ij} of Φ in (2.15) by $\sqrt{\varphi_{ii}\varphi_{jj}}$. That implies $\|\Delta A\|_{ij} \leqslant dd^T$ for d denoting the column vector with $d_k = \left(\frac{\varphi_{kk}A_{kk}}{1-\varphi_{kk}\mathbf{u}}\right)^{1/2}$ and the estimate $\|\Delta A\|_2 \leqslant \|dd^T\|_2 = d^T d$. Therefore

$$\|\Delta A\|_2 \leqslant \sum_{k=1}^{n} \frac{(\mu_k + 1)\mathbf{u}}{1 - (\mu_k + 1)\mathbf{u}} A_{kk}. \tag{2.19}$$

We later show numerical evidence that the new estimate (2.15) together with Corollary 2.9 improves upon the original one in [53, Lemma 2.2] by an order of magnitude and more, and upon (2.19) by about a factor 1.5. Executing the code in (2.18) in rounding upwards computes an upper bound for Φx because the quantities involved are positive.

3. Scaling, equilibration and approximation of smallest singular value.

Our verification method requires a Cholesky and/or LDL^T-decomposition of a symmetric matrix $A \in \mathbb{F}^{n\times n}$. To that end it is important to scale the matrix. Denote by $\kappa(A)$ the 2-norm condition number of A and by $\mathcal{D}_n$ the set of nonsingular diagonal $n \times n$ matrices. For Hermitian A an optimal diagonal scaling [6, Lemma 1] is symmetric

$$\inf_{D_1, D_2 \in \mathcal{D}_n} \kappa(D_1 A D_2) = \inf_{D \in \mathcal{D}_n} \kappa(DAD)\ .$$

If for positive definite A the diagonal is scaled to 1, then its condition number is at least not far from the optimal scaling by [54, Theorem 4.3]

$$\kappa(A) \leqslant q \min_{D \in \mathcal{D}_n} \kappa(D^H A D)$$

where q denotes the maximum number of nonzero elements per row of A. In order to avoid rounding errors by scaling we use

```
d = pow2(round(log2(1./sqrt(diagA))));  A = (d.*A).*d';
```

for symmetric positive definite A. Note that `d` is a vector. For D denoting the diagonal matrix with diagonal `d`, the command `(d.*A).*d'` is an efficient computation of DAD. No rounding errors occur because the elements of `d` are powers of 2. For a linear system $Ax = b$ we scale the right hand side by `b = d.*b`. If $\widehat{x}$ is the solution of the scaled linear system $DAD\widehat{x} = Db$, then $D\widehat{x}$ is the solution of the original linear system.

Practical experience suggests that an equilibration with $|A|$ being close to a scalar multiple of a doubly stochastic matrix is advisable [13, 1]. To that end the famous Sinkhorn-Knopp algorithm is the algorithm of choice. For a good introduction and historical remarks see [24]. For symmetric A a vector d is computed by the simple iteration `d = 1./(abs(A)*d)`. Starting with `d = ones(n,1)` it converges to a vector δ if, and only if, A has total support with $|DAD|$ being a scalar multiple of a doubly stochastic matrix for $D = \mathrm{diag}(\delta)$. In our case it is not necessary to compute δ with high accuracy because its entries are rounded to the nearest power of 2 to avoid rounding errors, and in our case a good starting vector for symmetric positive definite A is `1./sqrt(diag(A))`. We use 2 iteration steps, each scaling columns and rows:

$$\begin{aligned}&\mathtt{d = 1./sqrt(diagA);}\\ &\mathtt{for\ k = 1:4,\ d = 1./(abs(A)*d);\ end}\\ &\mathtt{A = (d.*A).*d';}\end{aligned} \tag{3.1}$$

For symmetric but indefinite A diagonal elements may be zero, so the scaling (3.1) is not applicable. Several scalings DAD are possible, for example using $D := \mathrm{diag}(d)$ with d_k being the columnwise maximum, or $\Sigma_\ell |A_{k\ell}|$. We use the Euclidean norm of columns together with the Sinkhorn-Knopp algorithm, i.e.,

$$\begin{aligned}&\mathtt{d = 1./vecnorm(A,2)';}\\ &\mathtt{for\ k = 1:4,\ d = 1./(abs(A)*d);\ end}\\ &\mathtt{A = (d.*A).*d';}\end{aligned} \tag{3.2}$$

The scaling of the right hand side and transformation of the solution is as before. For a general matrix we use Matlab's `equilibrate` and add two Sinkhorn-Knopp iterations [24]:

$$\begin{aligned}&\mathtt{[p,row,col] = equilibrate(A,'vector');}\\ &\mathtt{for\ k = 1:2}\\ &\quad\mathtt{col = 1./(abs(A(p,:)')*row);\ \ row = 1./(abs(A(p,:))*col);}\\ &\mathtt{end}\\ &\mathtt{row = sign(row).*pow2(round(log2(abs(row))));}\\ &\mathtt{col = sign(row).*pow2(round(log2(abs(col))));}\\ &\mathtt{A = row.*A(p,:).*col';}\end{aligned} \tag{3.3}$$

The outputs `p, row, col` of the function `equilibrate` are vectors. Denote the diagonal matrices with `row, col` in the diagonal by R, C, respectively, and the permutation matrix mapping $\{1, \ldots, n\}$ into `p` by P. Then the equilibrated matrix is $B := RPAC$ with entries close to ± 1 in the diagonal and all its off-diagonal entries limited by about 1 in absolute value. After transforming the right hand side into `c = row.*b(p,:)`, it follows $A^{-1}b = Cy$ for $By = c$. As in (3.2) we avoid rounding errors by replacing the entries of the vectors `row` and `col` by the nearest power of 2.

As has been mentioned we need two kinds of decompositions, Cholesky and LDL^T. Mathematically, pivoting is not necessary for symmetric positive input matrix A, however, permuting A may reduce the fill-in significantly. Therefore we use

$$\mathtt{[R,FLAG,p] = chol(A,'vector');} \tag{3.4}$$

producing an error flag and permutation information. For the permutation matrix P mapping $\{1,\dots,n\}$ into `p` it follows $R^TR \approx P^TAP$. The latter matrix is `A(p,p)` in Matlab notation.

Matlab offers two possibilities for scaling in the LDL^T-decomposition of a real symmetric matrix, both based on Duff's multifrontal method "MA57" [12]. First, a threshold for the pivot tolerance is introduced by the call

$$[\mathtt{L},\mathtt{D},\mathtt{p}] = \mathtt{ldl}(\mathtt{A},\mathtt{thresh},'\mathtt{vector}'); \tag{3.5}$$

such that LDL^T approximates `A(p,p)`. A larger threshold requires more computing time but may produce a more stable result. The maximum threshold is `thresh = 0.5`, and we always use this value.

There may be an obstacle when applying `ldl` to an augmented matrix $B := \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$. Here the blocks of D are all 2×2 with zero diagonal, see Lemma 9.1. In that case D should contain totally $2n$ nonzero entries for $A \in \mathbb{F}^{n\times n}$. However, it happens that (3.5) computes D with less nonzero elements, i.e., D is singular, even for moderate condition number. That happens when `ldl` is applied to the augmented matrix B and occurred in 54 out of 211 test cases. In such a case the part of L corresponding to zero blocks in D are the rows of the identity matrix. So a remedy may be to replace the zero blocks of D by the corresponding parts of `A(p,p)`. However, in that case the residual $LDL^T - A(p,p)$ is usually not small enough. Another remedy in that case $nnz(D) < n$ may be to use

$$[\mathtt{L},\mathtt{D},\mathtt{p}] = \mathtt{ldl}(\mathtt{A}+\mathtt{realmin}*\mathtt{speye(n)},\mathtt{thresh},'\mathtt{vector}');\;\; \mathtt{D}(1:\mathtt{n}+1:\mathtt{n}^2) = 0; \tag{3.6}$$

Then the factors L, D are practically unchanged by the tiny diagonal entries `realmin`, but that trick helps the algorithm to produce nonsingular D with diagonal entries of size `realmin`. The second statement sets the diagonal of D to zero so that all 2×2 blocks have zero diagonal - as it should be from the beginning. However, that may produce subnormal entries in L, and arithmetical operations including subnormal numbers are known to be slow. Thus we replace `realmin` by 10^{-50}:

$$\begin{aligned} &[\mathtt{L},\mathtt{D},\mathtt{p}] = \mathtt{ldl}(\mathtt{A}+\mathtt{1e}-\mathtt{50}*\mathtt{speye(n)},\mathtt{thresh},'\mathtt{vector}');\\ &\mathtt{D}(1:\mathtt{n}+1:\mathtt{n}^2) = 0;\\ &\forall i,j: \;\; |L_{ij}| \leqslant 10^{-30} \;\;\Rightarrow\;\; L_{ij} = 0 \end{aligned} \tag{3.7}$$

In our application it is safe to use the absolute shift by 10^{-50} because the input matrix has a norm close to 1. However, that trick may produce quite some fill-in, in particular with numbers very small in magnitude. Therefore we set in addition entries in L smaller than 10^{-30} in magnitude to zero. That reduces the fill-in significantly and still produces a factor L which is sufficiently accurate for our purposes.

Those tricks are necessary to cure the behaviour of Matlab's `ldl`. The reason is that $MA57$ [12] uses a "zero pivot tolerance" 10^{-20}. Unfortunately that applies not only to the entries of L but also to D, eventually producing a singular factor D. When changing the tolerance to zero, no singular factor D appears any more. In Matlab the user cannot change that tolerance. After reporting that behaviour to mathworks that may be possible in a future release and simplify our algorithms.

Beyond (3.5) a second possibility is an additional scaling using

$$[\mathtt{L},\mathtt{D},\mathtt{p},\mathtt{S}] = \mathtt{ldl}(\mathtt{A},\mathtt{thresh},'\mathtt{vector}');$$

In that case LDL^T approximates `S(p,:)*A*S(:,p)`. For our purposes the additional scaling was sometimes useful but often counterproductive. Therefore we compute throughout this note LDL^T-decompositions by (3.5), and if necessary by (3.7).

In our methods we need an approximation of the smallest singular value of some matrices. Since the matrices are large, `svd` is much too costly, and because they are sparse it should not be used anyway. One possibility is `svds(A,1,'smallestnz')`. That routine is fast, however, often pretty inaccurate.

In our applications we need approximations on $\sigma_{\min}(A)$ only for symmetric A. In that case we may use

$$\texttt{s} = \texttt{abs}(\texttt{eigs}(\texttt{A}, 1, '\texttt{smallestabs}')) \ . \tag{3.8}$$

Although the routine asks for the smallest absolute value of an eigenvalue, the result may be negative, therefore `abs(.)` is used as in [57]. That seems a stable and accurate method for symmetric input matrix, however, it is sometimes slow. Routine `eigs` is based on some iteration using some decomposition of A. In our applications we already have a decomposition, therefore we will compute $\tilde{s}(A,L) \lesssim \sigma_{\min}(A)$ by

$$\text{few inverse power iterations based on the factor } L \text{ of } A \ . \tag{3.9}$$

The result is multiplied by 0.9 to (hopefully) ensure that it is strictly less than $\sigma_{\min}(A)$. That is working well in our applications because A is symmetric.

Next we show how a lower bound for the smallest singular value of A is used to obtain entrywise and accurate error bounds for an approximation $\tilde{x}$ of $A^{-1}b$.

4. Error bounds for $A^{-1}b$ based on a lower bound for $\sigma_{\min}(A)$.

In the following sections we will derive individual methods to compute a lower bound of the smallest singular value of a symmetric positive definite, symmetric and general A. Those methods include a decomposition of A allowing for a fast computation of an approximate solution of $Ay = c$. We abbreviate this by $y = solve(A, c)$.

Entrywise error bounds for the solution $A^{-1}b$ are obtained by the approach in [59]. To further improve the accuracy we store an approximate solution as an unevaluated sum $\tilde{x} + \tilde{y}$. This technique was introduced in [44] and later called "staggered correction" [55]. Together with accurate dot products it often allows for almost maximally accurate error bounds.

```
1  $[\tilde{x}, \delta] = \text{ErrorBound}(A, b, s, \text{“}solve\text{“})$
2      $\tilde{x} = \text{solve}(A, b)$                                          $\% \ A^{-1}b \approx \tilde{x}$
3      $\tilde{y} = \text{solve}(A, [\![b - A\tilde{x}]\!]_{2,1})$                       $\% \ A^{-1}b \approx \tilde{x} + \tilde{y}$
4      $[\tilde{x}, \tilde{y}] = \text{TwoSum}(\tilde{x}, \tilde{y})$
5      $\tilde{z} = \text{solve}(A, [\![b - A\tilde{x} - A\tilde{y}]\!]_{2,1})$                 $\% \ A^{-1}b \approx \tilde{x} + \tilde{y} + \tilde{z}$
6      $[\tilde{x}, \tilde{y}] = \text{TwoSum}(\tilde{x}, \tilde{y} + \tilde{z})$                     $\% \ A^{-1}b \approx \tilde{x} + \tilde{y}$
7      setround(-1);  $\varrho = \text{abs}([\![A\tilde{x} + A\tilde{y} - b]\!]_{2,1})$
8      setround(+1);  $\varrho = \max(\varrho, \text{abs}([\![A\tilde{x} + A\tilde{y} - b]\!]_{2,1}))$
9      $\delta = |\tilde{y}| + \|\varrho\|_\infty / s$
```

Table 1
Residual iteration and inclusion of the solution $A^{-1}b$.

We sketch in Table 1 the rationale to compute accurate error bounds for $A^{-1}b$. From lines 2 and 3 it follows $\tilde{x} \approx A^{-1}b$ and $\tilde{y} \approx A^{-1}(b - A\tilde{x})$. Since the residual in the second line is calculated in extended precision, the unevaluated sum $\tilde{x} + \tilde{y}$ should be a good approximation to $A^{-1}b$. The fourth line[3] ensures that the bit patterns of $\tilde{x}$ and $\tilde{y}$ do not overlap. From line 5 the unevaluated sum $\tilde{x} + \tilde{y} + \tilde{z}$ improves the approximate solution further. The correction $\tilde{z}$ should be very small correcting the last bits of $\tilde{y}$. That is utilized in line 6. When computing

$$\begin{aligned} \varrho_1 &:= [\![A\tilde{x} - A\tilde{y} - b]\!]_{2,1} \quad \text{in rounding downwards} \\ \varrho_2 &:= [\![A\tilde{x} - A\tilde{y} - b]\!]_{2,1} \quad \text{in rounding upwards} \end{aligned}$$

it follows $\varrho_1 \leqslant A\tilde{x} - A\tilde{y} - b \leqslant \varrho_2$ and the ϱ in line 8 satisfies

$$|A\tilde{x} - A\tilde{y} - b| \leqslant \varrho \ .$$

Hence, proceeding as in [53] and abbreviating the vector of all $1's$ by $\mathbf{e}$ we obtain

$$\begin{aligned} |A^{-1}b - \tilde{x}| &= |\tilde{y} + A^{-1}(b - A\tilde{x} - A\tilde{y})| \\ &\leqslant |\tilde{y}| + \|A^{-1}\|_\infty \|\varrho\|_\infty \mathbf{e} \\ &\leqslant |\tilde{y}| + \|A^{-1}\|_2 \|\varrho\|_\infty \mathbf{e} \\ &= |\tilde{y}| + \sigma_{\min}(A)^{-1} \|\varrho\|_\infty \mathbf{e} \\ &\leqslant \delta \end{aligned} \tag{4.1}$$

because $s \leqslant \sigma_{\min}(A)$ and the computation of δ in the last line is in rounding upwards.

The residuals are computed using the extended precision package in [15] corresponding to a relative rounding error unit 2^{-113}. Therefore splitting the approximate solution into three parts $\tilde{x} + \tilde{y} + \tilde{z}$ would not improve the accuracy of the result. To that end we need higher precision for the computation of the residual. We show how to do that in Part II of this note.

Using accurate dot products is mandatory and ensures to obtain accurate entrywise error estimates. To see that we display in Table 2 the intermediate results for the residual iteration in Table 1 for two representative examples. The examples are number 1210 and 438 of [8], the first one being symmetric, the second one general. As we will see later neither our new algorithm `VerifySparselss` to be presented in Table 6 nor the algorithm in [57] could compute verified bounds for the first example 1210. The reason is that due to the condition number $1.2 \cdot 10^{15}$ both methods could not verify a lower bound for the smallest singular value.[4] This does not affect the iteration. We computed the smallest singular value using the multiple precision package [15] for the final bound in line 14 of Table 2.

The input is normed to $\|A\|_\infty = 1 = \|b\|_\infty$. The smallest singular value in line 4 of Table 2 shows that both matrices are ill-conditioned. Therefore we can expect that $\|\tilde{x}\|_2 \approx \|A^{-1}b\| \approx \|b\|/\sigma_{\min}(A) \approx \sigma_{\min}(A)^{-1}$ is large. That is certified in line 5, where $\tilde{x}$ is Matlab's $A\backslash b$. It is a well known fact in numerical analysis that, although the matrices are ill-conditioned, the residual norm $A\tilde{x} - b$ is small, and that is verified in line 6. The next line 7 displays the median and maximum of $|A^{-1}x - b|$. It is slightly better than expected by the well accepted rule of thumb that the error is of size $\mathbf{u} \cdot \text{cond}(A)$. That may be due to the sparseness of the input matrices.

[3]The call `[x,y] = TwoSum(a,b)` computes $x = \text{float}(a + b)$ for scalars, vectors and matrices a, b, and in addition y such that $x + y = a + b$ is mathematically correct [34].

[4]Our alternative method presented in Part II of this note succeeds to compute verified bounds.

Table 2
Detailed results for verified inclusion $A^{-1}b \in \tilde{x} \pm \delta$ by residual iteration

		symmetric		general	
1	# in [8]	1210		438	
2	n	20,360		1,633	
3	nnz(A)	509,866		46,626	
4	$\sigma_{\min}(A)$	$1.2 \cdot 10^{-15}$		$8.1 \cdot 10^{-12}$	
5	$\lVert\tilde{x}\rVert_\infty$	$1.7 \cdot 10^{11}$		$7.9 \cdot 10^{9}$	
6	$\lVert A\tilde{x} - b\rVert_\infty$	$7.1 \cdot 10^{-5}$		$3.9 \cdot 10^{-8}$	
7	error $\tilde{x}$	$3.9 \cdot 10^{-4}$	$3.9 \cdot 10^{-4}$	$1.5 \cdot 10^{-9}$	$2.4 \cdot 10^{-6}$
8	$\lVert A\tilde{x} + A\tilde{y} - b\rVert_\infty$	$5.2 \cdot 10^{-8}$		$3.1 \cdot 10^{-16}$	
9	error $\tilde{x} + \tilde{y}$	$3.1 \cdot 10^{-7}$	$3.1 \cdot 10^{-7}$	$3.0 \cdot 10^{-17}$	$2.0 \cdot 10^{-14}$
10	$\lVert A\tilde{x} + A\tilde{y} - b\rVert_\infty$	$4.3 \cdot 10^{-11}$		$5.2 \cdot 10^{-24}$	
11	error $\tilde{x} + \tilde{y}$	$2.4 \cdot 10^{-10}$	$2.4 \cdot 10^{-10}$	$2.0 \cdot 10^{-17}$	$5.4 \cdot 10^{-17}$
12	$\varrho = \lvert A\tilde{x} + A\tilde{y} - b\rvert$	$3.0 \cdot 10^{-15}$	$4.4 \cdot 10^{-11}$	$5.2 \cdot 10^{-26}$	$5.3 \cdot 10^{-24}$
13	$\delta = \lvert\tilde{y}\rvert + \varrho/s$	2.4	$3.5 \cdot 10^{4}$	$1.3 \cdot 10^{-8}$	$4.8 \cdot 10^{-7}$
14	entrywise accuracy of incl.	$1.5 \cdot 10^{-11}$	$2.1 \cdot 10^{-7}$	$4.1 \cdot 10^{-17}$	$1.1 \cdot 10^{-16}$

The next line in Algorithm ErrorBound in Table 1 improves $\tilde{x}$ by one step of residual iteration where the residual $A\tilde{x} - b$ is computed in extended and stored in working precision. The correction $\tilde{y}$ is not added to $\tilde{x}$, the approximate solution is kept as an unevaluated sum $\tilde{x} + \tilde{y}$. Line 4 in Algorithm ErrorBound in Table 1 makes sure that the bit representations of $\tilde{x}$ and $\tilde{y}$ do not overlap.

As shown in lines 8 and 9 of Table 2 the unevaluated sum $\tilde{x} + \tilde{y}$ has a smaller residual and better accuracy. By the cited rule of thumb the improvement should be of the order $\mathbf{u} \cdot \mathrm{cond}(A)$, in the second example it seems better.

Line 5 of Algorithm ErrorBound performs a second residual iteration based on the unevaluated sum $\tilde{x} + \tilde{y}$. The correction $\tilde{z}$ should be smaller than $\tilde{y}$ and is therefore added to $\tilde{y}$. For the new approximation $\tilde{x} + \tilde{y}$ line 6 ensures again that the bits don't overlap.

As by lines 10 and 11 in Table 2 this approximation has again smaller residual and improved accuracy. Correspondingly, the upper bound ϱ on $\lvert A\tilde{x} + A\tilde{y} - b\rvert$ is small, in the second example very small. Now the verified inclusion for $A^{-1}b$ consists of three parts, the approximation by the unevaluated sum $\tilde{x} + \tilde{y}$ and the normwise error bound $\alpha := \lVert\varrho\rVert_\infty / \sigma_{\min(A)}$, i.e., $\lvert A^{-1}b - (\tilde{x} + \tilde{y})\rvert \leqslant \alpha$.

By combining the error bound into the vector $\delta = \lvert\tilde{y}\rvert + \varrho/s$ this becomes an entrywise error bound $(A^{-1}b)_k \in \tilde{x}_k \pm \delta_k$. Note that δ is computed in rounding upwards in the last line of Algorithm "ErrorBound".

The last line in Table 2 shows the median and maximum accuracy of the inclusion in terms of the relative error $\lvert\delta_k/\tilde{x}_k\rvert$. In the first example at least 7 and on the median some 11 decimal figures of the left and right bounds coincide. In the second examples the bounds are maximally accurate, i.e., the bounds differ only in the last bit. Repeating the residual iteration in steps 5 and 6 of Algorithm ErrorBound in Table 1 another 3 times yields almost maximally accurate results for the first example as well, in the median and maximum.

5. Input data with tolerances. If the matrix and/or the right hand side are afflicted with tolerances, verified error bounds based on our methods can be computed as well. We give the details for real linear systems, for complex interval data an almost identical ansatz is applicable.

Consider $\mathbf{A} \in \mathbb{IF}^{n\times n}$ and $\mathbf{b} \in \mathbb{IF}^{n,k}$. The interval matrix $\mathbf{A} = [\underline{A}, \overline{A}]$ for $\underline{A}, \overline{A} \in \mathbb{F}^{n\times n}$ consists of all real matrices A with $\underline{A} \leqslant A \leqslant \overline{A}$ and similarly for $\mathbf{b}$. Then

$$\Sigma(\mathbf{A},\mathbf{b}) := \{x \in \mathbb{R}^{n\times k} : \exists A \in \mathbf{A}\ \exists b \in \mathbf{b} \text{ with } Ax = b\} \tag{5.1}$$

is sometimes called the "outer" solution set [37, 49]. In order to compute error bounds for $\Sigma(\mathbf{A},\mathbf{b})$ we use a midpoint-radius representation for $\mathbf{A}$. The INTLAB commands `mA = mid(A)` and `rA = rad(A)` compute matrices $mA, rA \in \mathbb{F}^{n\times n}$ with $mA - rA \leqslant A \leqslant mA + rA$ for all $A \in \mathbf{A}$, and similarly for $\mathbf{b}$.

For interval input, there is no need for an extra precise residual iteration as in Algorithm ErrorBound in Table 1. Denote by $\tilde{x}$ an approximate solution of the midpoint linear system $mA\cdot x = mb$ after few residual iterations. Denote $\check{A} := mA$ and $\widehat{\Delta} := rA$, and let $A \in \mathbf{A}, b \in \mathbf{b}$ fixed but arbitrary. Denote $\Delta := A - \check{A}$. Then $|\Delta| \leqslant |\widehat{\Delta}|$ and we adapt (4.1) into

$$\begin{aligned} |A^{-1}b - \tilde{x}| &= |(\check{A}+\Delta)^{-1}\left(b - \check{A}\tilde{x}\right)| \\ &= |(I + \check{A}^{-1}\Delta)^{-1}\check{A}^{-1}\left(b - \check{A}\tilde{x}\right)| \\ &\leqslant \frac{\|\check{A}^{-1}\|_\infty \|b - \check{A}\tilde{x}\|_\infty}{1 - \|\check{A}^{-1}\Delta\|_\infty}\,\mathbf{e} \\ &\leqslant \frac{\sigma_{\min}(\check{A})^{-1}\|\mathbf{b} - \mathbf{A}\tilde{x}\|_\infty}{1 - \sigma_{\min}(\check{A})^{-1}\|\widehat{\Delta}\|_\infty}\,\mathbf{e} \end{aligned} \tag{5.2}$$

which is true provided that $\sigma_{\min}(\check{A})^{-1}\|\widehat{\Delta}\|_\infty < 1$. Note that successful computation of a lower bound of $\sigma_{\min}(\check{A})$ verifies the nonsingularity of $\widehat{A}$ a posteriori. A larger diameter of $\mathbf{b}$ widens the bounds, a larger diameter of $\mathbf{A}$ reduces the range of applicability, i.e., verified bounds are only obtained for smaller condition number of $\check{A}$.

6. Symmetric (positive definite) matrices. As has been mentioned before, "positive definite" is in parenthesis because this is no assumption on the input matrix but will be proved a posteriori by our algorithm. As a consequence, the subalgorithm "verifySparseSPD" necessarily fails if the symmetric input matrix has nonpositive eigenvalues. In that case subalgorithm "verifySparseSym" will be called.

THEOREM 6.1. *Let symmetric $A \in \mathbb{F}^{n\times n}$ and $0 < s \in \mathbb{F}$ be given. For diagonal $D \in \mathbb{F}^{n\times n}$ assume $D_{kk} \geqslant s$ for all $k \in \{1,\ldots,n\}$. Suppose that the floating-point Cholesky decomposition of $B := A - D$ runs to completion producing a Cholesky factor $\tilde{R}$. Define $\Delta B := \tilde{R}^T\tilde{R} - B$. Then*

$$\sigma_{\min}(A) \geqslant s - \|\Delta B\|_2 \geqslant s - \|\Delta B\|_\infty. \tag{6.1}$$

Let $\mu \in \mathbb{N}^n$ with μ_i denoting the number of nonzero elements in the i-th column of $\tilde{R}$ and assume $\mathbf{u}\max\mu_k < 1$. Denote by $M \in \mathbb{R}^{n\times n}$ the matrix with $M_{ij} := \min(\mu_i,\mu_j) + 1$ and by $D \in \mathbb{R}^{n\times n}$ the diagonal matrix with $D_{kk} = \left(\frac{B_{kk}}{1 - M_{kk}\mathbf{u}}\right)^{1/2}$. Let

$$\alpha := \mathbf{u}\|DMD\|_2 \tag{6.2}$$

be as in Lemma 2.7 computed by Corollary 2.9. Assume $s \geqslant \alpha$. Then $\|\Delta B\|_2 \leqslant \alpha$ and

$$\sigma_{\min}(A) \geqslant s - \alpha \tag{6.3}$$

```
 1  function $[x,\delta]$ = verifySparseSPD(A,b)
 2      If any $A_{kk} \leqslant 0, [x,\delta]$ = verifySparseSym(A,b), return
 3      Equilibrate $A$ by (3.1)
 4      Compute Cholesky factorization $\tilde{R}^T\tilde{R} \approx A$ by (3.4)
 5      If failed, $[x,\delta]$ = verifySparseSym(A,b), return
 6      Compute $\tilde{s}(A,\tilde{R})$ by (3.9) and set $s := 0.9\tilde{s}$
 7      setround(-1); As = A - s * speye(n);
 8      setround(0); [Rs,FLAG,p] = chol(As);
 9      If succeeded, goto step 13
10      Set rounding downwards and $As = As + (8s/10)I;\ s = s/5;$
11      Compute Cholesky factor $\tilde{R}^T\tilde{R} \approx As$ in rounding to nearest by (3.4)
12      If failed, $[x,\delta]$ = verifySparseSym(A,b), return
13      Compute upper bound $\alpha :=$ r.h.s.(6.2) with $\|Rs^TRs - As\|_2 \leqslant \alpha$
14      If $\alpha \geqslant s$, compute $\alpha$ with $\|Rs^TRs - As\|_2 \leqslant \alpha$ using (2.11)
15      If $\alpha \geqslant s$, compute $\alpha$ with $\|Rs^TRs - As\|_2 \leqslant \alpha$ using (2.13)
16      If $\alpha \geqslant s$, verification failed, return
17      $[x,\delta]$ = ErrorBound$(A, b, s-\alpha,$ "*solve*"$)$ using $\tilde{R}$ for solve
```

TABLE 3
Verified error bounds for $A^{-1}b$ for symmetric positive definite sparse input matrix A.

if the decomposition was performed using nearest operations. If $\max \mu_k \leqslant (2\mathbf{u})^{-1/2}$, *then* (6.3) *remains true for faithful operations when replacing* $\mathbf{u}$ *in* (6.2) *by* $2\mathbf{u}$.

Proof. We have $\tilde{R}^T\tilde{R} = B + \Delta B$ with $\|\Delta B\|_2 \leqslant \alpha$ by Lemma 2.7. Moreover, ΔB being symmetric implies $\|\Delta B\|_2 \leqslant \|\Delta B\|_\infty$. Hence (1.10) yields

$$\lambda_{\min}(A) - s \geqslant \lambda_{\min}(A - D) = \lambda_{\min}(B) = \lambda_{\min}(\tilde{R}^T\tilde{R} - \Delta B) \geqslant -\|\Delta B\|_2 \geqslant -\alpha$$

and proves $\lambda_{\min}(A) \geqslant s - \alpha \geqslant 0$, and therefore (6.3). The assertion for faithful operations follows as in Lemma 2.7. ∎

In Table 3 we sketch our subalgorithm "verifySparseSPD" for solving a sparse linear system with symmetric positive definite matrix. More precisely, the algorithm assumes only that the input matrix A is symmetric. If A is indefinite and/or positive definiteness cannot be verified, then our subalgorithm "verifySparseSym" for symmetric input matrix as given in the next section is called.

The details of subalgorithm "verifySparseSPD" are as follows. If there are nonpositive diagonal elements of A the matrix cannot be positive definite and we call subalgorithm "verifySparseSym". Otherwise, after equilibration in line 3 a numerical Cholesky decomposition `[R,FLAG,p] = chol(A,'vector')` is computed in line 4. If `FLAG` $\neq 0$, the factorization failed and subalgorithm "verifySparseSym" is called. Otherwise, an approximative lower bound s of the smallest singular value of A is computed in line 6.

In line 7 a lower bound `As` of the shifted matrix $A - sI$ is computed. Hence `As` $= A - D$ with $D_{kk} \geqslant s$ and Theorem 6.1 is applicable. Next, a floating-point Cholesky decomposition of `As` is tried in line 8. In case of failure we try again with a smaller value for s. In the actual implementation we avoid using two matrices but set `As = As - s*speye(n)` in line 10. It needs some care to use the correct matrix `As` with the updated s. Denote the matrix `As` in line 8 by $\widehat{As}$. From line 7 and rounding downwards we know $\widehat{As} = A - D$ for diagonal D with $D_{kk} \geqslant s$. Denote `s' = 8*s/10` in rounding downwards and the new s computed at the end of line 10 by $\overline{s}$. Note that $\overline{s} \leqslant s/5$. Then rounding downwards implies $s' \leqslant 0.8s$ and $\widehat{As}$ is updated in line 10 into some $\overline{As} := \widehat{As} + \widehat{D} = A - D + \widehat{D}$ for diagonal $\widehat{D}$ with $\widehat{D}_{kk} \leqslant s' \leqslant 0.8s$. Note that $\overline{As}$ is the matrix `As` after executing step 10. It follows $D_{kk} - \widehat{D}_{kk} \geqslant s - 0.8s = s/5 \geqslant \overline{s}$ so that the new $\widehat{As}$ in line 10 is equal to $A - \tilde{D}$ for diagonal $\tilde{D}$ with $\tilde{D}_{kk} \geqslant \overline{s}$. Thus Theorem 6.1 and (6.2) are applicable for $\overline{As}, \overline{s}$.

The decomposition in line 11 may fail because of ill-conditioned input matrix A or, if s is chosen too large. In that case we call subalgorithm "verifySparseSym". In the next line 13 an upper α as in (6.2) in Theorem 6.1 is computed using the code in Corollary 2.9 such that (using rounding downwards) $s-\alpha$ is a lower bound of $\sigma_{\min}(A)$. This first upper bound on α comes by (6.2) at practically no cost. If α is too large, i.e., $\alpha \geqslant s$, we compute $\Delta B := Rs^T Rs - As$ in rounding downwards and upwards and improve α by initializing `setround(1), Q = Rs'*Rs-As;` and using (2.11) in Lemma 2.4. If still $\alpha \geqslant s$, we improve α again by computing ΔB in extended precision with rounding to nearest and using (2.13). Step 14 could be omitted, however, if successful it saves quite some computing time.

This is our last resource. It still $\alpha \geqslant s$, subalgorithm "verifySparseSPD" failed to compute verified error bounds. In that case our general Algorithm `verifySparselss` to be presented in Table 6 calls subalgorithm "verifySparseSym". Otherwise, $s - \alpha$ rounded downwards is a correct lower bound for the smallest singular value of A, and an improved approximate solution x together with error bound δ satisfying $A^{-1}b \in x \pm \delta$ is computed by Algorithm "ErrorBound" in Table 1. This algorithm requires to solve a linear system $Ay = c$ for some right hand side c which is performed using $\tilde{R}$ in the fourth line.

7. Factorization of 2×2 Hermitian matrix.

Let L and D be factors of a real symmetric or Hermitian matrix A such that $A = LDL^H$. Then D comprises of 1×1 or 2×2 real symmetric or Hermitian blocks, respectively. Let B be such a block matrix. We will factor $B = MSPM^H$ with symmetric or Hermitian M, possibly complex signature matrix S and permutation matrix P such that $\text{cond}(M) \approx \text{cond}(B)^{1/2}$.

The purpose is as follows. Applying the factorization to the blocks of D results in a block factorization $D = \widehat{M}\widehat{S}\widehat{P}\widehat{M}^H$. Setting $L_1 := L\widehat{M}\widehat{S}\widehat{P}$ and $L_2 = L\widehat{M}$ yields $A = L_1 L_2^H$. Since S and P are unitary, the sets of singular values of L_1 and L_2 are identical. It follows $\text{cond}(A) \leqslant \text{cond}(L_1)^2 = \text{cond}(L_2)^2$. Although, in contrast to the Cholesky decomposition, the condition number of L_1 (and L_2) is, in general, not equal to $\text{cond}(A)^{1/2}$, practical evidence suggests that they are often not too far apart.

For the anticipated decomposition we distinguish three cases. If B is 1×1, then $B = b$ for a real or complex number b, and $M := \sqrt{|b|}$, $S = \text{sign}(b)$ and $P = 1$ do the job.

The second case is a 2×2 matrix with zero diagonal, i.e., $B := \begin{pmatrix} 0 & b \\ \bar{b} & 0 \end{pmatrix}$. In that case we choose

$$M := \begin{pmatrix} \sqrt{|b|} & 0 \\ 0 & \sqrt{|b|} \end{pmatrix}, \quad S := \begin{pmatrix} \text{sign}(b) & 0 \\ 0 & \text{sign}(\bar{b}) \end{pmatrix} \quad \text{and} \quad P := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

For the third case let nonsingular Hermitian $B = \begin{pmatrix} a & b \\ \bar{b} & c \end{pmatrix}$ be given and define $d :=$ $\sqrt{(a-c)^2 + 4\bar{b}b}$. Its (real) eigenvalues are $\lambda_{1,2} = \frac{1}{2}(a + c \pm d)$, and for $b \neq 0$ the unitary eigenvectors are $v_{1,2} = \begin{pmatrix} a - c \pm d \\ 2\bar{b} \end{pmatrix}$. It follows the eigendecomposition $B = VDV^H$ for unitary $V := \begin{pmatrix} v_1/\|v_1\|_2 & v_2/\|v_2\|_2 \end{pmatrix}$ and $D := \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$. Hence

$$M := V \begin{pmatrix} \sqrt{|\lambda_1|} & 0 \\ 0 & \sqrt{|\lambda_2|} \end{pmatrix}, \quad S := \begin{pmatrix} \operatorname{sign}(\lambda_1) & 0 \\ 0 & \operatorname{sign}(\lambda_2) \end{pmatrix} \quad \text{and} \quad P = I$$

is the desired decomposition.

In the first two cases we just need $\sqrt{|b|}$. The third case looks also like a straightforward approach, and in almost all cases it worked well. However, for b being small in absolute value compared to a and/or c numerical problems may occur. We come to that when discussing the test results in Section 12.

Summarizing we showed that for an LDL^T-decomposition of a real symmetric matrix A the block diagonal matrix D can be expressed as

$$D = \widehat{D}S\widehat{D}^T \qquad \text{for symmetric } A \tag{7.1}$$

$$D = \widehat{D}SP\widehat{D}^T \qquad \text{for symmetric } A \text{ with zero diagonal} \tag{7.2}$$

with block diagonal symmetric $\widehat{D}$, real signature matrix S and permutation matrix P. If A is complex, then $D = \widehat{D}SP\widehat{D}^H$, $\widehat{D}$ is block diagonal Hermitian and S is a complex signature matrix.

8. Symmetric matrices.

We show in Table 4 a general outline of our subalgorithm "verifySparseSym" to compute verified bounds for the solution of a sparse linear system with symmetric matrix.

After equilibration in line 2 we decompose A in line 3. It occurs very rarely that D is singular; in that case we call[5] the subalgorithm "verifySparseGen". It happened during testing, but not in our test suite of 48 symmetric test cases. Otherwise L_1, L_2 are computed in lines 5 – 6 with $A \approx L_1L_2$. The factors are computed in floating-point, but because S is a signature matrix the multiplication $L_2 := SL_1^T$ is error-free in floating-point. Thus, the factors L_1, L_2 have identical sets of singular values. Hence (1.11) gives

$$\sigma_{\min}(A) \approx \sigma_{\min}(L_1L_2) \geqslant \sigma_{\min}(L_1)\sigma_{\min}(L_2) = \sigma_{\min}(L_1)^2 = \sigma_{\min}(L_1L_1^T) \ . \tag{8.1}$$

Next $M = \text{float}(L_1L_1^T)$ is computed in line 7 in rounding upwards, that is $L_1L_1^T \leqslant M$, and in line 8 we use an approximation of $\sigma_{\min}(M)$ as an anticipated lower bound $\tilde{s} \lesssim \sigma_{\min}(A)$ on the smallest singular value of A. We approximate $\sigma_{\min}(M)$ because a Cholesky decomposition of M shifted by s is used in line 15 to compute a true lower bound on $\sigma_{\min}(M)$ leading to a lower bound for $\sigma_{\min}(A)$.

For a correct lower bound on $\sigma_{\min}(A)$ we compute an upper bound α on $\|A - L_1L_2\|_2$ in line 9. If α is not small enough, i.e., $\alpha \geqslant s$, then α is improved by (2.11) in line 10. Next we use (2.9) to compute an upper bound β on $\|M - L_1L_1^T\|_2$. Here

[5]Here the original data A, b before the equilibration in line 2 is to be used.

```
1  function $[x,\delta]$ = verifySparseSym(A,b)
2      Equilibrate $A$ by (3.2)
3      Compute $LDL^T(A)$ by (3.5)
4      If $D$ is singular, verification failed, $[x,\delta]$ = verifySparseGen(A,b), return
5      Compute approximate splitting $D \approx \widehat{D}S\widehat{D}^T$ according to (7.1)
6      Compute $L_1 \approx L\widehat{D}$ and $L_2 = SL_1^T$
7      Compute $M \approx L_1L_1^T$ in rounding upwards
8      Compute Compute $\tilde{s}(M,L_1)$ by (3.9) and set $s := 0.9\tilde{s}$
9      Use (2.10) to compute $\alpha$ with $\|A - L_1L_2\|_2 \leqslant \alpha$
10     If $\alpha \geqslant s$, improve $\alpha$ as in (2.11)
11     If $\alpha < s$, use (2.9) to compute $\beta$ with $\|M - L_1L_1^T\|_2 \leqslant \beta$, else $\beta = \infty$
12     If $\alpha < s$ and $\alpha + \beta \geqslant s$, improve $\beta$ as in (2.11)
13     If $\alpha + \beta \geqslant s$, recompute $M$ and improve $\alpha, \beta$ as in (2.13)
14     If $\alpha + \beta \geqslant s$, verification failed, $[x,\delta]$ = verifySparseGen(A,b), return
15     Compute $\widehat{M} := M - sI$ in rounding downwards
16     Compute Cholesky factor $\tilde{R}^T\tilde{R} \approx \widehat{M}$ in rounding to nearest by (3.4)
17     If succeeded, goto step 20
18     Set rounding downwards and $\widehat{M} = \widehat{M} + (8s/10)I$; $s = s/5$;
19     Compute Cholesky factor $\tilde{R}^T\tilde{R} \approx \widehat{M}$ in rounding to nearest by (3.4)
20     If failed, $[x,\delta]$ = verifySparseGen(A,b), return
21     Compute $\gamma$ with $\|\widehat{M} - \tilde{R}^T\tilde{R}\|_2 \leqslant \gamma$ by (6.2) in rounding upwards
22     If $\alpha + \beta + \gamma \geqslant s$, improve $\gamma$ as in (2.11)
23     If $\alpha + \beta + \gamma \geqslant s$, improve $\gamma$ as in (2.13)
24     If $\alpha + \beta + \gamma \geqslant s$, verification failed, $[x,\delta]$ = verifySparseGen(A,b), return
25     $[x,\delta]$ = ErrorBound($A, b, s - \alpha - \beta - \gamma$, "*solve*") using $LDL^T$ for solve
```

TABLE 4
Verified error bounds for $A^{-1}b$ for symmetric sparse input matrix A.

$\mathbf{u}$ in (2.9) is to be replaced by $2\mathbf{u}$ because M was computed in rounding upwards in line 7. Thus $L_1L_1^T \leqslant M$. If β is too large, i.e., if $\alpha + \beta \geqslant s$, then one additional matrix multiplication suffices to improve β as in (2.11) by computing `R = L1*L1'-M; beta = NormBnd(R,true)` in rounding downwards. This is true because the computation of M and $\mathrm{R} \leqslant L_1L_1^T - M$ imply $0 \leqslant M - L_1L_1^T \leqslant -R$.

If still $\alpha + \beta \geqslant s$, then we try in line 13 to further improve the error bounds. First we improve α by using (2.13). For β we use (2.13) as well, where this includes the recomputation of M in rounding to nearest. We refrain from recomputing s for the new M because numerical evidence suggests that, if any, a potential improvement is marginal. If still $\alpha + \beta \geqslant s$, then the verification failed and subalgorithm "verifySparseGen" will be called.

In line 15 the shifted matrix $\widehat{M}$ is computed in rounding downwards so that Theorem 6.1 is applicable. Next a floating-point Cholesky decomposition of $\widehat{M}$ is tried in line 16. If not successful, $\widehat{M}$ and s are updated as in lines 10−12 of "verifySparseSPD", and for the smaller shift s a Cholesky decomposition is tried in line 19.

If the second decomposition is still not successful, then the verification failed and subalgorithm "verifySparseGen" will be called. Otherwise, an upper bound γ from the right hand side in (6.2) is computed in line 21 satisfying $\|\widehat{M}-\tilde{R}^T\tilde{R}\|_2 \leqslant \gamma$. If necessary, γ is improved using (2.11) or (2.13). Now Theorem 6.1 implies $\sigma_{\min}(M) \geqslant s-\gamma$.

If the sum $\alpha+\beta+\gamma$ of errors is too large, then the verification failed and we turn to subalgorithm "verifySparseGen". Otherwise, i.e., $\alpha+\beta+\gamma<s$, (1.10), (8.1) and Theorem 6.1 yield

$$\begin{aligned} \sigma_{\min}(A) &\geqslant \sigma_{\min}(L_1L_2) - \|A-L_1L_2\|_2 \geqslant \sigma_{\min}(L_1L_1^T) - \|A-L_1L_2\|_2 \\ &\geqslant \sigma_{\min}(M) - \|L_1L_1^T - M\|_2 - \|A-L_1L_2\|_2 \geqslant \sigma_{\min}(M) - \beta - \alpha \\ &\geqslant s-\alpha-\beta-\gamma\ . \end{aligned} \tag{8.2}$$

Hence $\alpha+\beta+\gamma<s$ verifies that the matrix A is nonsingular, and entrywise bounds for the solution are computed by Algorithm ErrorBound in Table 1.

9. General matrices.

As in [48, 57] our method for linear systems with general matrix uses the augmented matrix

$$B := \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix} \tag{9.1}$$

the singular values of which are ± the eigenvalues of A. So in principle we could apply the methods for symmetric input matrix described in Section 8. However, due to the structure of the augmented matrix B the decomposition part is simpler as by the following lemma.

Lemma 9.1. *For nonsingular $A \in \mathbb{R}^{n\times n}$ a block LDL^T-decomposition of the augmented matrix B in* (9.1) *produces D with all diagonal elements being zero, i.e., D consists only of* 2×2 *pivot blocks with zero diagonal.*

Remark 9.2. There may exist LDL^T-decompositions of B with D having nonzero diagonal entries. For the 1×1 matrix $A=1$ the augmented matrix B is a permutation matrix, and a computation yields that all LDL^T-decompositions satisfy $L = \begin{pmatrix} 1 & 0 \\ \varphi & 1 \end{pmatrix}$ and $D = \begin{pmatrix} 0 & 1 \\ 1 & -2\varphi \end{pmatrix}$ for some $\varphi\in\mathbb{R}$. That includes the block LDL^T-decomposition obtained by $\varphi=0$.

Proof. A block LDL^T-decomposition is based on [14, Section 11.1]

$$PBP^T = \begin{pmatrix} E & C^T \\ C & G \end{pmatrix} = \begin{pmatrix} I_s & 0 \\ CE^{-1} & I_{n-s} \end{pmatrix} \begin{pmatrix} E & 0 \\ 0 & G-CE^{-1}C^T \end{pmatrix} \begin{pmatrix} I_s & E^{-1}C^T \\ 0 & I_{n-s} \end{pmatrix}$$

with I_m denoting the $m\times m$ identity matrix and $s\in\{1,2\}$. The diagonal of the augmented matrix B remains zero under symmetric permutations, so that the first pivot must be 2×2 with $E = \begin{pmatrix} 0 & \alpha \\ \alpha & 0 \end{pmatrix}$. Moreover, $\begin{pmatrix} E & C^T \end{pmatrix}$ comprises of the k-th and m-th row of B for some $1\leqslant k\leqslant n$ and $n+1\leqslant m\leqslant 2n$. Let P be the permutation matrix mapping $(1,\dots,2n)$ to $(k,m,1,\dots,k-1,k+1\dots,m-1,m+1,\dots,2n)$. Then G

is square with $2n-2$ rows and columns and has the same structure as the augmented matrix in (9.1). Hence the structure of PBP^T is described by

$$\begin{pmatrix} E & C^T \\ C & G \end{pmatrix} = \begin{pmatrix} 0 & \alpha & 0_{1,n-1} & v^T \\ \alpha & 0 & u^T & 0_{1,n-1} \\ 0_{n-1,1} & u & 0_{n-1,n-1} & H^T \\ v & 0_{n-1,1} & H & 0_{n-1,n-1} \end{pmatrix}$$

with column vectors $u, v \in \mathbb{R}^{n-1}$, a square matrix H with $n-1$ rows and columns, and 0 denoting a matrix of zeros with dimension according to the subscripts. Then

$$CE^{-1}C^T = \alpha^{-1} \begin{pmatrix} 0_{n-1,1} & u \\ v & 0_{n-1,1} \end{pmatrix} \begin{pmatrix} u^T & 0_{1,n-1} \\ 0_{1,n-1} & v^T \end{pmatrix} = \alpha^{-1} \begin{pmatrix} 0_{n-1,n-1} & uv^T \\ vu^T & 0_{n-1,n-1} \end{pmatrix}$$

shows that $G - CE^{-1}C^T$ has the same structure as the augmented matrix (9.1). The result follows. ∎

In contrast to [46, 48, 57] we proceed for general matrices as follows. After equilibrating the original matrix A we compute an LDL^T-decomposition of the augmented matrix B by (3.5). The permutation information for pivoting is stored in the vector p such that $B(p,p) \approx LDL^T$. According to Lemma 9.1 the matrix D has exactly $2n$ nonzero entries for nonsingular A. If the decomposition fails, i.e., there are less than $2n$ nonzero elements in D, we use LDL^T-decomposition as in (3.7). As has been mentioned that happened in 54 out of 211 test cases.

A splitting (7.2) of D is computed, and in lines 7 and 8 the factors L_1, L_2 such that $L_1L_2 \approx B(p,p)$. The factor L_2 is L_1 multiplied by some signature and permutation matrix. That computation is error-free, so that as in subalgorithm "verifySparseSym" the factors L_1, L_2 have identical sets of singular values. Hence (8.1) is true when replacing A by B or $B(p,p)$.

The first bound on α is computed in line 11 using (2.10). In line 5 of that code `NormBnd(C,false)` is used and `C` should be replaced by `B`. In fact, `NormBnd(B,true)` could be used. However, we use `NormBnd(A,false)` because the spectral norms of A and B coincide but A has half the size of B.

The remaining of the subalgorithm until line 20 is identical to subalgorithm `VerifySparseSym` in Table 4, so that (1.10), (8.1) and Theorem 6.1 yield

$$\begin{aligned} \sigma_{\min}(A) &= \sigma_{\min}(B) \geqslant \sigma_{\min}(L_1L_2) - \|B - L_1L_2\|_2 \geqslant \sigma_{\min}(L_1L_1^T) - \|B - L_1L_2\|_2 \\ &\geqslant \sigma_{\min}(M) - \|L_1L_1^T - M\|_2 - \|B - L_1L_2\|_2 \geqslant \sigma_{\min}(M) - \beta - \alpha \\ &\geqslant s - \alpha - \beta - \gamma\ . \end{aligned}$$

Error bounds for the solution of the original linear system $Ax = b$ use that

$$(9.2) \qquad \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ b \end{pmatrix}$$

implies $x = A^{-1}b$. The residual iteration in Algorithm `ErrorBound` is adapted to the augmented system, and the lower bound $s - \alpha - \beta - \gamma$ for $\sigma_{\min}(A) = \sigma_{\min}(B)$ and the LDL^T-decomposition from line 4 or 5 is used for the residual iteration. The approximation x with error bound δ refers to the first n entries of the result of "ErrorBound".

```
1  function $[x,\delta]$ = verifySparseGen(A,b)
2      Equilibrate $A$ by (3.3)
3      Let $B$ the augmented matrix (9.1)
4      Compute $LDL^T(B)$ by (3.5)
5      If $nnz(D) < 2n,$ compute $LDL^T(B)$ by (3.7)
6      If $nnz(D) < 2n,$ verification failed, return
7      Compute approximate splitting $D \approx \widehat{D}SP\widehat{D}^T$ according to (7.2)
8      Compute $L_1 \approx L\widehat{D}$ and $L_2 = SPL_1^T$
9      Compute $M \approx L_1L_1^T$ in rounding upwards
10     Compute $\tilde{s}(M,L_1)$ by (3.9) and set $s := 0.9\tilde{s}$
11     Use (2.10) to compute $\alpha$ with $\|B - L_1L_2\|_2 \leqslant \alpha$
12     If $\alpha \geqslant s,$ improve $\alpha$ as in (2.11)
13     If $\alpha < s,$ use (2.9) to compute $\beta$ with $\|M - L_1L_1^T\|_2 \leqslant \beta,$ else $\beta = \infty$
14     If $\alpha < s$ and $\alpha + \beta \geqslant s,$ improve $\beta$ as in (2.11)
15     If $\alpha + \beta \geqslant s,$ recompute $M$ and improve $\alpha, \beta$ as in (2.13)
16     If $\alpha + \beta \geqslant s,$ verification failed, return
17     Compute $\widehat{M} := M - sI$ in rounding downwards
18     Compute Cholesky factor $\tilde{R}^T\tilde{R} \approx \widehat{M}$ in rounding to nearest by (3.4)
19     If succeeded, goto step 23
20     Set rounding downwards and $s = 8s/10;$ $\widehat{M} = \widehat{M} + sI;$ $s = s/5;$
21     Compute Cholesky factor $\tilde{R}^T\tilde{R} \approx \widehat{M}$ in rounding to nearest by (3.4)
22     If Cholesky decomposition ends premature, verification failed, return
23     Compute $\gamma$ with $\|\widehat{M} - \tilde{R}^T\tilde{R}\|_2 \leqslant \gamma$ by (6.2) in rounding upwards
24     If $\alpha + \beta + \gamma \geqslant s,$ improve $\gamma$ as in (2.11)
25     If $\alpha + \beta + \gamma \geqslant s,$ improve $\gamma$ as in (2.13)
26     If $\alpha + \beta + \gamma \geqslant s,$ verification failed, return
27     $[x,\delta]$ = ErrorBound$(B, [0;b], s - \alpha - \beta - \gamma,$ "solve") using $LDL^T$ for solve
```

TABLE 5
Verified error bounds for $A^{-1}b$ for general sparse input matrix A.

10. Complex sparse linear systems and the first sparse lss algorithm.

Unfortunately, the LDL^T-decomposition for sparse matrices in Matlab is restricted to real data. For a complex linear system $(A + iB)(x + iy) = b + ic$ a simple remedy is to use the augmented linear system

$$\begin{pmatrix} A & -B \\ B & A \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix} \tag{10.1}$$

of doubled size. Then for positive definite Hermitian, for Hermitian and for general matrix $A + iB$ the augmented matrix $C := \begin{pmatrix} A & -B \\ B & A \end{pmatrix}$ is symmetric positive definite, symmetric, and general, respectively. In each case the singular values of C are those

```
function [xs,delta] = verifySparselss(A,b)
% Approximate solution xs of Ax=b with error bound delta
  if isreal(A)
    if isreal(b)            % A and b real
      symm = isequal(A',A);
      if symm               % A symmetric
        [xs,delta] = verifySparseSPD(A,b);
      end
      if ( ~symm ) || isnan(xs(1))   % A unsymm. or SPD failed
        [xs,delta] = verifySparseGen(A,b);
      end
    else                    % A real, b complex
      [xs,delta] = verifySparselss(A,[real(b) imag(b)]);
      n = size(A,1);
      m = size(b,2);
      xs = complex(xs(:,1:m),xs(:,m+1:end));
      delta = reshape(vecnorm(reshape(delta,[],2),2,2),n,[]);
    end
  else                      % A complex
    n = size(A,1);
    A = [real(A) -imag(A);imag(A) real(A)];
    b = [real(b);imag(b)];
    [xs,delta] = verifySparselss(A,b);
    xs = complex(xs(1:n,:),xs(n+1:end,:));
    delta = reshape(delta,n,[])';  % take care of multiple r.h.s.
    delta = reshape(vecnorm(reshape(delta,2,[]),2),size(b,2),[])';
  end
end  % function verifySparselss
```

Table 6

Algorithm to compute verified error bounds for the solution of a sparse linear system.

of $A + iB$ doubled, so that the condition number does not change. A drawback is that for general matrices we use the augmented matrix (9.1) resulting in a linear system of four times the dimension of the original complex system. If a complex LDL^T-decomposition will be included in Matlab, then that drawback disappears.

In the previous sections we described subalgorithms to compute error bounds for the solution of linear systems with symmetric positive definite matrix, with symmetric and with general matrix. For a given linear system we may check symmetry, but positive definiteness may not be known beforehand. Therefore, we present in Table 6 the self-contained Algorithm `verifySparselss` as executable Matlab/INTLAB code to solve a general real or complex sparse linear systems. The final and also a second version of Algorithm `verifySparselss` including least squares problems and underdetermined linear systems will be given in Table 8 in Part II of this note.

The algorithm proceeds as follows. First it is checked for real or complex data. If the matrix is complex, error bounds are computed according to (10.1), if only b is complex it suffices to solve a linear systems with 2 right hand sides. In either case the error bound is the entrywise Euclidean norm of the bounds for the real and complex part.

If the input matrix A is symmetric, subalgorithm “verifySparseSPD” is tried. If the check of positivity of all diagonal elements of A or some Cholesky decomposition fails, then “verifySparseSPD” calls subalgorithm “verifySparseSym”. If it fails as well, then as a final resource subalgorithm “verifySparseGen” is called. If the input matrix is not symmetric, then subalgorithm “verifySparseGen” is called immediately.

The subalgorithms cover multiple right hand sides for real and complex input data. For complex b and/or A some care is necessary to collect the data for the complex inclusion.

We refrain from giving an explicit algorithm for data afflicted with tolerances because it is clear how to proceed along the lines given in Section 5.

11. Comparison of the new algorithm and [57].

For a linear system $Ax = b$ the algorithm proposed by Terao and Ozaki [57] is based on Theorem 1.1 to compute a lower bound for $\sigma_{\min}(A)$, basically as in Table 7.

If successful, i.e., $\theta > \varrho$, then $\sigma_{\min}(B) = \sigma_{\min}(A) > \theta - \varrho$. The Matlab code is published in [57] and some missing code was kindly provided by the authors. In [57] the quality of an inclusion was improved by a residual iteration based on

$$\begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} A^T b \\ b \end{pmatrix} \tag{11.1}$$

with solution $y = b$ and $x = A^{-1}b$. The advantage of their method compared to

1	Apply (3.8) to B as in (9.1) and set $\theta := 0.5s$
2	Compute $LDL^T(B + \theta I)$ by (3.5)
3	If the inertia of D is not $(n, 0, n)$, decrease θ and go to step 2
4	Compute ϱ with $\|B + \theta I - LDL^T\|_2 \leqslant \varrho$
5	If $\theta \leqslant \varrho$, restart from step 2 with larger $\theta > \varrho$ or verification fails

TABLE 7
Computation of a lower bound $\theta - \sigma$ for $\sigma_{\min}(A)$.

Theorem 1.1 in [46] is that only one decomposition, namely of $B + \theta I$ is necessary because for nonsingular A the inertia $(n, 0, n)$ of B is known beforehand. The trade-off is that only a decomposition of the shifted matrix $B + \theta I$ is available, not of B. It was analysed in [53] that nevertheless a residual iteration with this decomposition converges, i.e., improves the solution of (11.1), and this is used in [57]. Suppose $LDL^T = B + \theta I$ and $\widehat{L}\widehat{D}\widehat{L}^T = B$. If A is well-conditioned, then θ is large introducing a significant difference between L, D and $\widehat{L}, \widehat{D}$. If A is ill-conditioned, then θ is small but the factors are sensitive to perturbations of B. Nevertheless a residual iteration using the factors L, D converges [53], but more iterations are necessary compared to using the original factors $\widehat{L}, \widehat{D}$ of B.

A second difficulty is that an inclusion of the product of three matrices is needed in step 4. In [57] the code

```
[L,D,p] = ldl(mid(G),'vector');  rho = NormBnd(G(p,p) - L * intval(D) * L',true);
```

computes ϱ with $\|B + \theta I - LDL^T\|_2 \leqslant \varrho$ and uses `NormBnd` from (1.9). The first product `M := L*intval(D)` is an inclusion of LD, so that the product ML^T of an interval matrix times point matrix causes additional overestimation. That reduces the maximal possible condition number until which a verification is possible.

A third problem slowing down [57] is that the decomposition of the shifted matrix B causes significantly more fill-in than the decomposition of the original augmented matrix B. We come to that in Part II of this note.

The algorithm in [57] is called by

$$\texttt{X} = \texttt{verifylinsys}(\texttt{A}, \texttt{b}, \texttt{precond}, \texttt{acc}) \tag{11.2}$$

with additional parameters `precond` and `acc`. The output `X` is an interval vector, and if successful, $A^{-1}b \in \texttt{X}$. The meaning of `acc` is as follows. When multiplying two interval matrices, there is a choice in INTLAB [47] between using midpoint-radius arithmetic and rank-1 updates. The former produces bounds which are slightly wider for small radii of the factors, but for very large radii up to a factor 1.5 wider than those of the latter. However, interval matrix multiplication using the midpoint-radius representation is much faster than using rank-1 updates [50]. To choose either method the commands `intvalinit('FastIVmult')` and `intvalinit('SharpIVmult')` are used. If `acc` is *true*, then the slower method eventually producing better bounds is activated.

However, the two approaches differ only if both factors comprise of intervals with nonzero diameter. The most important product in the code of [57] in Table 7 is `L*intval(D)*L'`, but here always one factor is a point matrix. Therefore there is no difference between the two methods in INTLAB for multiplication. Consequently, we observed a marginal difference between the quality of the bounds using $false$ or *true* for `acc`, which is confirmed by the test results in [57]. Therefore, the computational results in the next section use `acc` = $false$.

If the extra parameter `precond` is *true*, then before executing the code in Table 7 the equilibration as in (3.3) is applied. Switching `precond` on or off has significant influence on the performance and accuracy of the algorithm in [57]. In many cases `precond` = *true* both reduces the computing time and increases the accuracy significantly, and often verification fails without preconditioning. Rarely we observed failure of verification with and success without preconditioning. In our computational results we found 3 such cases and appended the computing time by an "*".

Another reason to use `precond` = *true* for the algorithm in [57] is that when using `precond` = $false$ the inclusion may be wide. For instance, in example 1404 the verified inclusion by [57] with `precond` = $false$ ends successfully, but all entries of the inclusion are equal to $[-4.45 \cdot 10^{17}, 4.45 \cdot 10^{17}]$.

12. Test results. Our computing environment is a Panasonic laptop CF-SV with Intel(R) Core(TM) i7-10810U CPU with 1.10/1.61 GHz and 16 GB RAM. We use Matlab version 2023b [33] under Windows 10.

As for test matrices we used the Suite Sparse Matrix Collection [8] with the interface [21]. More precisely, we took all real and complex square matrices with dimension

$$10^3 \leqslant n \leqslant 10^5 \quad \text{and} \quad 10^{10} \leqslant \texttt{condest(A)} \leqslant 10^{16} \quad \text{and} \quad \texttt{nnz(A)} \leqslant 10^6 \ . \tag{12.1}$$

That resulted in totally 306 tests displayed in Table 8. The first column indicates the structure indicated by [8], namely symmetric positive definite, symmetric indefinite, general real, all test matrices out of [57], complex Hermitian positive definite and general complex. Our Algorithm `verifySparselss` computed verified bounds in 301 out of the 306 real and complex test cases. In the 302 real test cases satisfying (12.1) were 26 examples where [57] failed to compute verified bounds in all four combinations of options `precond` and `acc`. In all those 26 examples `verifySparselss` succeeded. We found no example vice versa, i.e., `verifySparselss` failed but [57] succeeds in some combination. However, there are surely such cases.

TABLE 8
Test sets and success rate.

structure	success new			success [57]		
spd	22	out of	22	14	out of	22
sym	45	out of	48	42	out of	48
gen	210	out of	211	199	out of	211
[57]	20	out of	20	20	out of	20
complex spd	1	out of	1			
complex gen	3	out of	4			

We compare our algorithm to that in [57], and also against Matlab's "backslash" operator, henceforth depicted by `lu`. The latter provides an approximate solution whereas our Algorithm `verifySparselss` and [57] deliver error bounds which are, although computed in floating-point, correct with mathematical certainty. Moreover, we try to provide close to maximally accurate bounds, i.e., the left and right bound of all entries should differ by few bits. Nevertheless, in some 37 out of the 306 test cases our Algorithm `verifySparselss` is faster than `lu`. That should never happen because the verified bounds are an approximation with error bound. That confirms once again that there is hardly a panacea, i.e., a general purpose algorithm to solve sparse linear systems. In the median `lu` is 4.9 times faster than `verifySparselss`.

The dimension, number of nonzero elements and condition number of all test cases is shown in Figure 1. The dimensions vary between 1019 and $682,862$ and the number of nonzero elements between 3562 and $5,778,545$. For given matrix of dimension n we generate a right hand side `A*(2*rand(n,1)-1))` so that the solution has, up to rounding errors, uniformly distributed entries between -1 and 1. In [57] the right hand side `A*ones(n,1)` was used.

In [57] computational results are listed for the four options `acc` and `precond` *true* and $false$, but no clear recommendation was given which combination to use. In order to display a fair comparison we proceed as follows. As noted above there is practically no difference in choosing *true* or $false$ for `acc`. It remains the choice for `precond`. As *true* is mostly superior, we first try to compute verified bounds by (11.2) with `precond` = *true* and `acc` = $false$. If successful, the computing time and accuracy for this setting is reported. If not successful, we try again with both `precond` and `acc` set to $false$. If now successful, the computing time and accuracy for this setting is reported. That is indicated in our listings by an "*" after the computing time of [57]. There are 3 such cases in the test suite satisfying (12.1), namely numbers $430, 46$ and 1395 in [8]. If still not successful, the minimum of the computing time (to realize failure) for the two settings is reported together with NaN for the accuracy indicating that the verification failed.

In Figure 2 we show for all tests the ratio of computing times of `lu` divided by that for our new Algorithm `verifySparselss` (henceforth abbreviated by "new"), and for the algorithm in [57] divided by "new". The ratios in the left graph are displayed if "new" is successful, i.e., computes verified error bounds, and the ratios in the right graph are displayed if both "new" and [57] are successful. That explains some gaps.

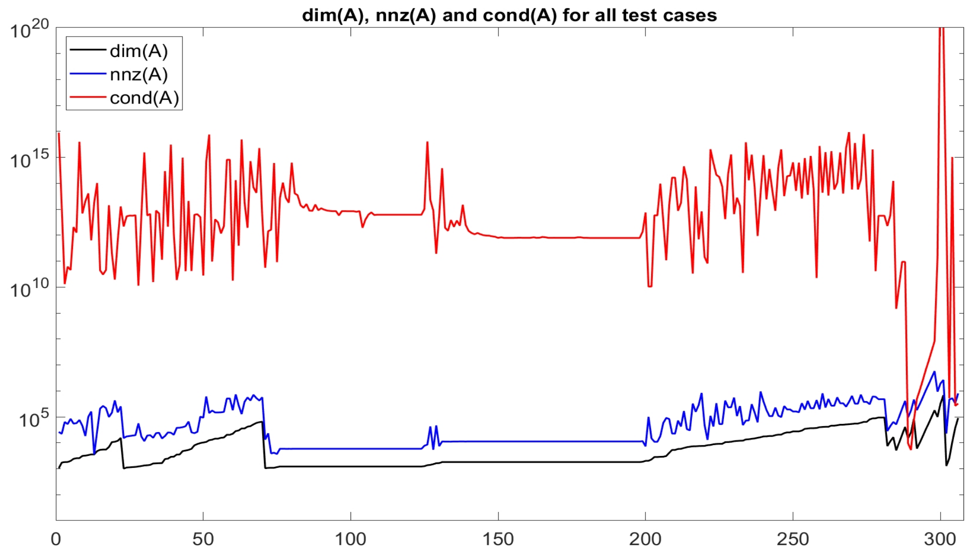

FIG. 1. *Dimension, number of nonzero elements and condition number of all test matrices.*

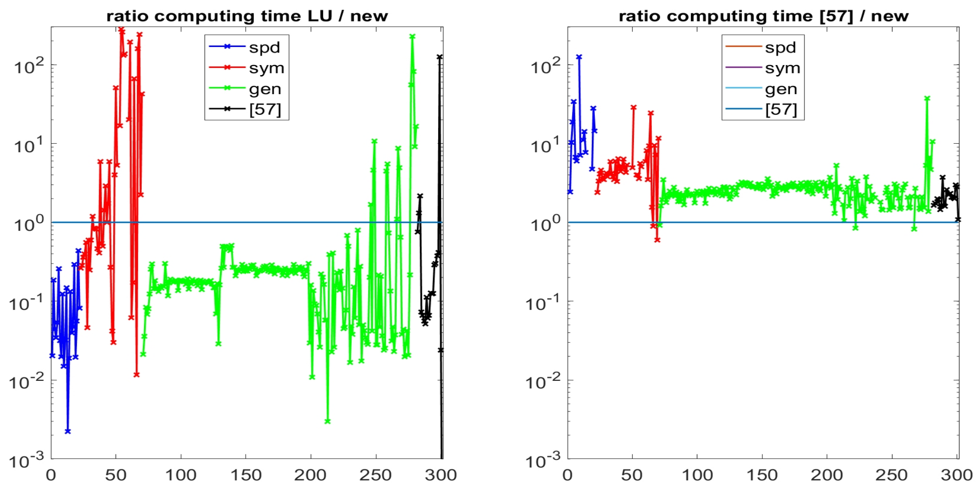

FIG. 2. *Ratios of computing times $t_{\texttt{lu}}/t_{new}$ and $t_{[57]}/t_{new}$.*

A number less than 1 in the left graph means that `lu` is faster than “new”, and a number larger than 1 in the right graph means that “new” is faster than [57]. In the median over all examples `lu` is faster than “new” by a factor 4.9. But in 9 out of the 306 cases `lu` is slower than “new” by 2 orders of magnitude, e.g. in number 2214 in [8] by a factor 281, in example 2231 “new” is 261 times faster than `lu`. In the first case the number of nonzero elements of the factor L in our algorithm is $430,688$, whereas `lu` produces factors L, U with $16,300,793$ and $47,932,779$ elements, respectively. That may explain the large computing time. Neither reverse Cuthill-McKee nor minimum degree reordering changes the situation for `lu`.

In some 4 cases the maximum relative error of the approximation by `lu` exceeds 0.01, i.e., at most 2 figures of some entries of the approximation are correct. Depending on the right hand side, the maximal relative error to the true solution $A^{-1}b$ may exceed 1, i.e., some entries of the approximation computed by `lu` have a wrong sign.

TABLE 9
The 5 *best and worst time ratios* $t_{[57]}/t_{new}$ *out of the* 301 *real test cases*

matrix		*times* [*sec*]		*relerr new*		*relerr* [57]		
#	n	t_{new}	$t_{[57]}$	*median*	*max*	*median*	*max*	$t_{[57]}/t_{new}$
1306	62500	588.025	348.852	3.7e-17	8.2e-15	1.7e-14	1.6e-5	0.59
1414	49702	11.859	9.686	3.7e-17	1.8e-13	1.4e-14	4.9e-8	0.82
2814	8256	6.249	5.278	3.7e-17	2.3e-16	1.6e-12	5.1e-7	0.84
2536	43887	5.319	4.761	3.7e-17	3.7e-15	9.7e-15	1.2e-9	0.89
838	1048	0.152	0.140	3.5e-17	1.1e-16	1.8e-15	5.1e-13	0.92
39	10974	0.637	17.794	3.5e-17	1.1e-16	2.8e-15	3.1e-11	27.91
2221	10798	6.837	195.825	3.7e-17	1.1e-16	2.4e-13	1.8e-7	28.64
35	2003	0.156	5.330	3.7e-17	1.1e-16	3.7e-15	7.8e-12	34.10
1374	87190	6.985	263.287	3.7e-17	1.8e-15	6.1e-15	8.9e-9	37.70
45	3134	0.117	14.708	3.6e-17	1.1e-16	2.6e-15	3.5e-11	125.62

In the median our new method is faster than [57] by a factor 2.7. In all but 5 of the test cases "new" was faster than [57]. In Table 9 we list the 5 test cases with smallest ratio $t_{[57]}/t_{new}$ of computing times and the 5 test cases with the largest ratio.

In the worst case "new" is 1.7 times slower than [57]. That is number 1306 in [8], where the matrix has dimension $62,500$ with $424,966$ nonzero elements and an estimated condition number $2.3{\cdot}10^{15}$. The computing time for `lu` is 1304 seconds, the new algorithm needs 588 seconds to compute verified bounds with maximal entrywise relative error $8.2{\cdot}10^{-15}$. For that example [57] computes verified bounds with maximal relative error $1.6 \cdot 10^{-5}$ in 349 seconds.

Next we show in Figure 3 a rough image of the median relative error of the approximation by `lu` and of the verified bounds of "new" and [57]. The relative error of "new" is often not far from maximal accuracy so that we can use the bounds to compute the relative error of the approximation by `lu`. As can be seen `lu` computes usually approximations with some 13 correct figures, but sometimes only few figures are correct. In the median the inclusions by [57] are usually accurate to about 15 correct figures.

We discuss some details of our Algorithm **`verifySparselss`** in Table 6 on the several improvement steps in the subalgorithms "verifySparseSPD", "verifySparseSym" and "verifySparseGen". As has been mentioned our first priority is the successful computation of verified bounds, and to that end there are several measures in the subalgorithms to avoid failure. Secondly, we aim to compute highly accurate bounds. One might introduce options to change these priorities.

We start with "verifySparseSPD" which is called if the input matrix is symmetric. If this subalgorithm fails, then "verifySparseSym" is called. Therefore, "verifySparseSPD" can only fail in step 16 if $\alpha \geqslant s$. That was not the case in all 22 examples in spd in Table 8. Hence, the Cholesky factorizations in steps 4 of A and in step 11 of the shifted matrix where all successful. The upper bound α on the residual of the Cholesky factors in step 13 was improved as in (3.9) using Perron-Frobenius Theory. In the median some 6 power iterations were used for the spd examples. The

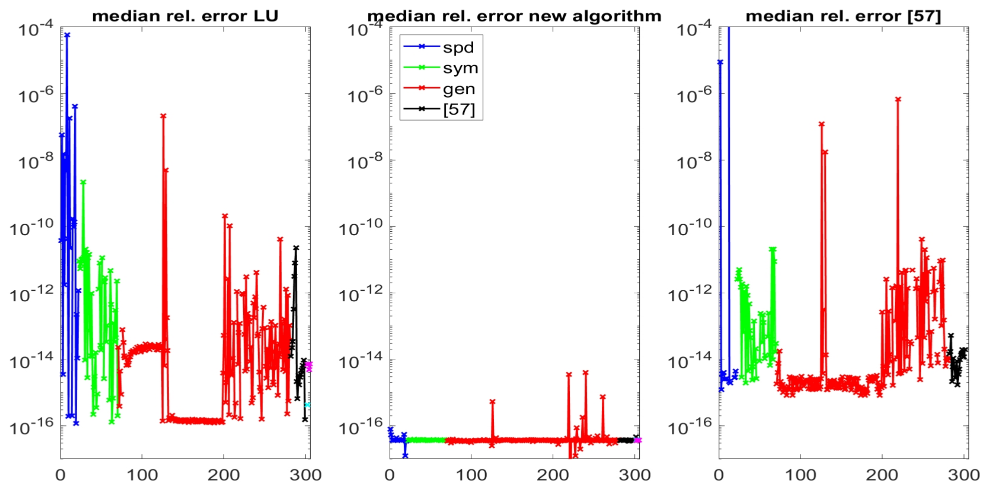

FIG. 3. *Median of relative errors of* `lu`*, by the new algorithm and [57].*

first improvement of α in step 14 was used in 3 out of the 22 examples, the second improvement in line 15 was never necessary.

Next we discuss subalgorithm “verifySparseSym”. The security measure on singular D in step 4 occurred occasionally while developing Algorithm `verifySparselss`, in the sym tests with (12.1) it did not happen. The improvement of α in line 10 was used in 8 out of the 48 tests in sym, i.e., in the remaing 40 the a priori bound (2.10) was sufficient. The improvement of β in line 12 was used in 5 out of the 48 tests in sym, and the improvement of α and β in step 13 was used in 6 cases. Failure in line 14 occurred in 4 out of the 48 sym tests and Algorithm `verifySparselss` called subalgorithm “verifySparseGen”. The reason seems that subalgorithm “verifySparseGen” performs an unsymmetric equilibration by (3.3). The Cholesky decomposition in line 16 failed in 2 cases implying the computation of a new value of s in steps $18-19$, and “verifySparseSym” ended successfully with the new s. The bound γ required in the median some 7 iterations (3.9) in line 21. The improvement of γ in line 22 was used in 7 cases which were, with one exception, the same as for the improvement of α in line 10, the second improvement of γ in line 23 was used once in the 48 sym tests.

Subalgorithm “verifySparseSym” failed in 4 out of 48 cases and Subalgorithm “verifySparseGen” was called. In two of those cases, namely matrix 1210 and 1451 in [8], numerical difficulties in the splitting of D in Step 5 according to (7.1) occurred. In both cases the initial α in Step 7 was $1.4 \cdot 10^{-3}$ with no improvement in step 12. This is far too large for the anticipated lower bound $\tilde{s} = 3.9 \cdot 10^{-13}$ of $\sigma_{\min}(M)$. The reason is the poor splitting of D implying that $\|A(p,p) - L_1L_2\|_1 = 1.4 \cdot 10^{-3}$ is much larger than $\|A(p,p) - LDL^T\|_1 = 1.3 \cdot 10^{-10}$ for the LDL^T-decomposition in (3.5).

A remedy is to compute the splitting of D according to (7.1) in some higher precision. Since these are few operations it would not take much computing time. Then $\|A(p,p) - L_1L_2\|_1 = 1.6 \cdot 10^{-10}$ if not far from $\|A(p,p) - LDL^T\|_1$ as expected, the first approximation of α is $1.0 \cdot 10^{-8}$ in Step 9, with a final improvement in Step 13 into $\alpha = 1.8 \cdot 10^{-10}$. This is not enough for a successful verification but shows that in the two examples 1210 and 1451 the poor splitting of D was part of the problem.

TABLE 10
Timing and accuracy for sparse linear systems in [8] satisfying the conditions in Table 12.1, in particular $t_{[57]}/t_{new} > 1.84$ for all other tests not shown.

	matrix				*times [sec]*			*relerr* `lu`		*relerr new*		*relerr* [57]		
# *matrix*		n	$nnz(A)$	`condest(A)`	$t_{\texttt{lu}}$	t_{new}	$t_{[57]}$	*median*	*max*	*median*	*max*	*median*	*max*	$t_{[57]}/t_{new}$
spd	358	1050	26198	9.0e15	0.007	0.325	0.184	3.7e-11	8.7e-1	7.9e-17	1.7e-11	NaN	NaN	
	430	1733	22189	1.2e13	0.013	0.071	0.172*	5.6e-8	7.3e-4	5.3e-17	3.0e-14	8.7e-6	8.0e-3	2.43
	411	2568	75628	4.0e15	0.004	0.185	3.857	5.8e-5	2.3e-1	4.3e-17	3.0e-12	NaN	NaN	
	440	3363	99471	4.1e13	0.005	0.143	2.627	1.8e-7	1.1e-4	3.7e-17	1.0e-15	NaN	NaN	
	46	3562	159910	6.4e11	0.037	0.250	2.766*	2.2e-11	7.9e-7	3.9e-17	1.1e-16	2.7e4	3.6e12	11.07
	1611	5357	207123	4.4e10	0.050	0.375	9.180	1.7e-10	2.2e-6	3.8e-17	1.1e-16	NaN	NaN	
	1607	5489	262943	3.1e10	0.018	0.453	24.431	9.5e-11	2.3e-5	3.8e-17	1.1e-16	NaN	NaN	
	1610	5489	217669	4.6e10	0.016	0.341	14.076	1.3e-10	9.9e-5	3.7e-17	1.2e-16	NaN	NaN	
	413	6867	98671	1.4e13	0.059	0.203	2.440	4.1e-7	3.3e-3	5.5e-17	6.6e-14	NaN	NaN	
	47	15439	252241	1.3e13	0.045	0.552	16.886	1.2e-12	6.6e-6	3.7e-17	1.1e-16	NaN	NaN	
sym	2412	8034	23626	6.2e12	0.036	0.870	0.578	1.3e-12	2.4e-5	3.6e-17	3.9e-16	NaN	NaN	
	2410	9000	26556	6.4e12	0.048	1.615	0.747	7.8e-12	1.3e-3	3.5e-17	1.1e-16	NaN	NaN	
	1560	9769	101635	5.0e12	1.768	0.443	2.034	2.7e-14	5.0e-6	3.7e-17	1.1e-16	NaN	NaN	
	1247	12546	140034	7.6e15	8.761	20.856	2.712	?	?	NaN	NaN	NaN	NaN	
	1210	20360	509866	8.1e14	0.252	1009.215	171.385	?	?	NaN	NaN	NaN	NaN	
	1451	20360	509866	8.1e14	0.220	1009.785	170.518	?	?	NaN	NaN	NaN	NaN	
	949	41731	559341	1.9e12	129.853	131.650	204.919	1.2e-13	4.4e-7	3.7e-17	1.1e-16	2.1e-11	3.0e-5	1.56
	2536	43887	426898	7.2e14	0.062	5.319	4.761	3.3e-14	1.0e-7	3.7e-17	3.7e-15	9.7e-15	1.2e-9	0.89
	1306	62500	424966	2.3e15	1303.534	588.025	348.852	2.2e-12	5.0e-4	3.7e-17	8.2e-15	1.7e-14	1.6e-5	0.59
gen	838	1048	13299	5.6e10	0.003	0.152	0.140	2.3e-14	1.2e-8	3.5e-17	1.1e-16	1.8e-15	5.1e-13	0.92
	243	1080	23094	1.4e12	0.010	0.280	0.452	3.9e-16	7.6e-11	3.6e-17	1.1e-16	4.0e-15	1.9e-10	1.62
	1057	1220	5892	2.7e13	0.005	0.040	0.072	7.9e-14	8.7e-5	3.4e-17	2.0e-16	2.1e-15	4.9e-12	1.79
	1081	1220	5892	8.4e12	0.006	0.034	0.086	2.2e-14	4.2e-5	3.2e-17	1.1e-16	1.3e-15	2.5e-13	2.50
	214	1374	8588	4.1e15	0.012	0.083	0.175	2.1e-7	3.9e-2	5.4e-16	4.6e-10	1.2e-7	2.8e-2	2.10
	438	1633	46626	1.9e11	0.011	0.375	1.379	5.0e-9	2.7e-3	3.7e-17	2.4e-16	NaN	NaN	
	893	1650	7419	5.6e12	0.007	0.042	0.094	1.8e-13	1.4e-4	4.4e-17	4.6e-13	1.7e-8	9.5e0	2.23
	546	2880	18229	9.7e13	0.022	0.316	0.549	2.7e-12	3.9e-7	3.5e-17	1.1e-16	2.6e-12	1.5e-7	1.73
	465	2904	58142	3.5e12	0.016	0.401	0.522	2.1e-16	8.2e-10	2.7e-17	1.1e-16	6.6e-15	5.5e-9	1.30

TABLE 11
Timing and accuracy for sparse linear systems in [8] satisfying the conditions in Table 12.1, in particular $t_{[57]}/t_{new} > 1.84$ for all other tests not shown.

		matrix			*times [sec]*			*relerr* `lu`		*relerr new*		*relerr* [57]		
# matrix		n	$nnz(A)$	`condest(A)`	$t_{\texttt{lu}}$	t_{new}	$t_{[57]}$	*median*	*max*	*median*	*max*	*median*	*max*	$t_{[57]}/t_{new}$
gen	414	5773	71701	8.8e12	0.054	0.934	3.684	1.1e-14	1.9e-5	3.6e-17	1.1e-16	NaN	NaN	
	548	5850	42568	1.8e13	0.058	1.001	1.686	1.3e-13	2.0e-7	3.5e-17	1.1e-16	1.4e-12	5.9e-7	1.68
	818	6316	167178	4.5e14	0.004	1.318	1.384	1.8e-16	9.9e-12	3.7e-17	1.1e-16	1.6e-15	5.9e-11	1.05
	934	7055	30082	1.7e12	0.023	1.275	0.709	?	?	NaN	NaN	NaN	NaN	
	739	7337	156508	7.6e13	0.583	1.431	2.295	7.0e-15	1.3e-4	4.2e-17	2.6e-10	1.6e-12	7.3e-3	1.60
	1395	7548	834222	8.3e12	2.829	18.047	65.110*	1.2e-13	3.2e-7	3.5e-15	6.5e-9	6.8e-7	7.4e-3	3.61
	2814	8256	109368	2.1e15	0.865	6.249	5.278	9.6e-13	2.0e-4	3.7e-17	2.3e-16	1.6e-12	5.1e-7	0.84
	580	9129	52883	1.7e14	0.100	2.219	3.039	5.6e-14	2.3e-8	3.7e-17	1.1e-16	4.9e-13	9.9e-8	1.37
	741	10672	232633	2.3e14	1.560	2.268	3.335	5.1e-15	2.3e-3	4.1e-17	9.9e-9	4.7e-12	6.1e-1	1.47
	743	10964	233741	1.3e15	1.804	3.654	4.395	2.4e-13	2.4e-5	8.7e-17	2.0e-5	4.8e-12	8.6e-1	1.20
	415	12005	259577	1.3e13	0.108	2.825	8.905	8.6e-15	1.5e-4	3.6e-17	4.3e-16	NaN	NaN	
	745	14270	307858	1.3e15	4.006	5.054	4.256	5.0e-13	1.6e-3	1.8e-16	7.2e-6	NaN	NaN	
	756	15606	61484	9.4e11	0.284	1.029	0.766	7.7e-13	6.4e-6	3.7e-17	5.4e-16	NaN	NaN	
	922	16428	948696	4.2e13	3.712	213.325	896.348	3.3e-13	1.9e-8	3.6e-17	2.1e-16	NaN	NaN	
	747	17576	381975	1.2e15	6.407	127.736	7.250	4.1e-12	1.5e-4	4.0e-15	7.6e-4	NaN	NaN	
	553	17730	183325	2.4e13	0.534	12.651	22.897	5.0e-14	1.5e-7	3.5e-17	1.1e-16	1.3e-12	2.6e-6	1.81
	582	18289	106803	3.6e14	0.229	6.685	9.773	5.7e-14	1.5e-7	3.7e-17	1.5e-16	2.6e-12	2.7e-5	1.46
	431	19716	227872	8.8e12	0.117	4.123	6.254	1.4e-15	1.7e-4	4.6e-17	1.1e-11	NaN	NaN	
	572	20614	111903	3.9e14	0.660	3.266	7.734	1.1e-15	7.6e-7	3.8e-17	3.8e-12	1.3e-12	3.2e-2	2.37
	1109	25187	193276	1.9e14	12.804	2.803	4.014	3.7e-13	1.1e-7	3.6e-17	1.1e-16	4.1e-11	1.2e-5	1.43
	584	27449	160723	6.4e14	0.345	12.325	19.022	9.0e-14	7.6e-8	3.7e-17	3.6e-16	3.9e-12	1.8e-5	1.54
	574	27534	151063	6.3e14	1.017	4.695	14.400	2.7e-15	1.2e-6	4.3e-17	1.4e-10	2.0e-11	3.2e-1	3.07
	576	34454	190224	9.4e14	1.381	6.654	19.392	1.4e-15	1.8e-6	4.9e-17	6.5e-8	1.1e-11	9.9e0	2.91
	586	36609	214643	1.1e15	0.438	18.316	27.089	9.2e-14	1.9e-7	3.7e-17	1.1e-13	4.3e-12	1.0e-3	1.48
	578	41374	229385	1.6e15	1.483	11.079	22.503	1.7e-15	1.1e-6	7.4e-16	3.1e-3	NaN	NaN	
	588	45769	268563	1.7e15	1.142	24.244	36.506	1.0e-13	1.2e-6	3.7e-17	4.0e-11	5.5e-12	4.4e-1	1.51
	1413	49702	333029	1.5e15	106.792	96.786	8.472	8.2e-16	2.9e-3	4.6e-17	4.0e-11	NaN	NaN	
	983	51993	380415	9.4e15	77.666	120.593	476.677	4.1e-11	1.7e-4	2.9e-17	1.1e-16	NaN	NaN	

TABLE 12
Timing and accuracy for sparse linear systems in [8] satisfying the conditions in Table 12.1, in particular $t_{[57]}/t_{new} > 1.84$ for all other tests not shown.

	matrix				*times [sec]*			*relerr* `lu`		*relerr new*		*relerr* [57]		
# *matrix*		n	$nnz(A)$	`condest(A)`	$t_{\mathtt{lu}}$	t_{new}	$t_{[57]}$	*median*	*max*	*median*	*max*	*median*	*max*	$t_{[57]}/t_{new}$
gen	590	54929	322483	3.6e15	1.376	31.044	47.146	1.2e-13	3.9e-7	3.7e-17	3.6e-8	8.9e-12	1.3e2	1.52
	592	64089	376395	7.9e15	1.586	37.501	57.911	1.6e-13	1.1e-5	3.7e-17	9.7e-7	9.5e-12	3.8e4	1.54
	2657	87936	593276	4.1e10	1843.310	8.021	11.059	8.3e-13	3.6e-8	3.6e-17	1.1e-16	1.1e-14	4.5e-10	1.38
TO	917	7500	28462	2.3e12	0.303	0.401	0.661	1.2e-14	1.1e-9	3.5e-17	1.1e-16	1.4e-14	1.0e-10	1.65
	918	11532	44206	5.9e12	1.021	0.773	1.369	2.2e-14	3.9e-10	3.6e-17	1.1e-16	1.7e-14	4.6e-10	1.77
	919	16428	63406	1.2e14	2.934	1.349	2.305	3.4e-14	3.2e-5	3.6e-17	3.7e-15	5.3e-14	5.0e-6	1.71
	2564	5108	51412	1.5e9	0.018	0.253	0.505	3.2e-13	1.1e-9	3.7e-17	1.1e-16	2.1e-15	4.0e-12	2.00
	2565	10228	102876	1.1e10	0.036	0.541	1.045	3.1e-12	7.2e-9	3.7e-17	1.1e-16	5.9e-15	1.1e-11	1.93
	2566	20468	206076	9.4e10	0.072	1.280	1.849	7.9e-12	4.1e-7	3.7e-17	1.1e-16	1.1e-14	4.5e-10	1.44
	2567	40948	412148	9.4e10	0.149	2.877	4.640	2.3e-11	5.5e-7	3.7e-17	1.1e-16	5.6e-15	1.2e-10	1.61
	288	14734	95053	9.6e3	0.169	1.499	5.586	2.1e-15	1.3e-11	3.7e-17	1.1e-16	2.8e-15	3.8e-11	3.73
	289	25228	175027	5.3e3	0.246	4.044	6.615	6.5e-16	8.4e-12	3.7e-17	1.1e-16	5.5e-15	8.3e-11	1.64
	290	84617	463625	7.5e4	0.592	8.935	14.250	3.2e-15	6.4e-10	3.7e-17	1.1e-16	8.7e-15	4.4e-10	1.59
	2820	6005	182168	4.9e5	0.087	0.682	1.781	1.7e-15	3.5e-11	3.5e-17	1.1e-16	1.7e-15	8.4e-11	2.61
	2821	10142	312814	1.1e6	0.176	1.421	3.231	2.8e-15	2.2e-10	3.5e-17	1.1e-16	3.5e-15	2.0e-9	2.27
	2822	17922	561677	2.6e6	0.380	3.012	6.851	3.5e-15	2.3e-8	3.5e-17	1.1e-16	5.1e-15	1.5e-8	2.27
	2823	32510	1030878	6.3e6	1.841	6.349	14.839	4.7e-15	3.8e-8	3.6e-17	1.1e-16	9.6e-15	4.4e-8	2.34
	2824	56021	1797934	1.4e7	4.271	14.121	29.359	6.5e-15	2.0e-9	3.6e-17	1.1e-16	1.3e-14	2.2e-8	2.08
	2825	100037	3226066	3.4e7	11.940	31.849	65.795	6.4e-15	1.7e-7	3.7e-17	1.1e-16	1.5e-14	2.1e-7	2.07
	2826	178437	5778545	8.2e7	32.825	78.873	158.310	9.5e-15	1.3e-6	3.7e-17	1.1e-16	1.9e-14	1.1e-6	2.01
	1415	99340	940621	1.5e11	1254.259	9.978	29.648	1.5e-16	9.7e-10	3.7e-17	1.1e-16	1.1e-14	9.4e-10	2.97
	1417	321821	1931828	5.1e22	memory	18.346	52.045			3.7e-17	8.4e-16	1.2e-14	1.7e-8	2.84
	1419	682862	2638997	9.5e19	crash	302.435	326.737			4.7e-17	1.9e-8	1.9e-14	8.7e-5	1.08
cspd	1621	1280	22778	6.0e12	0.013	0.123	-	4.2e-16	6.1e-11	3.9e-17				
cgen	326	2534	463360	5.2e5	0.040	11.599	-	7.0e-15	7.8e-11	3.6e-17				
	1407	10605	522387	1.0e15	9.410	80.998	-	?	?	NaN				
	2555	37365	330633	2.7e5	0.451	30.738	-	4.9e-15	6.3e-11	3.7e-17				
	2556	90249	803173	3.2e5	1.520	105.123	-	7.1e-15	5.1e-10	3.7e-17				

The computation of the splitting of D in some higher precision would not require not much computing time, however, those problems seem rare, and in the two cases where they occurred the more precise splitting of D was still not enough for a successful verification. Therefore we refrained from changing our algorithm in that regard.

Finally, some details on the performance of subalgorithm "verifySparseGen" for the 211 "gen" test cases plus the 20 tests from [57]. The second LDL^T-decomposition in step 5 was necessary in 54 out of 231 cases due to singularity of the factor D. There seems room for improvement for the Matlab routine `ldl` for an augmented matrix of type (9.1) with zero diagonal. With the trick in (3.7) the LDL^T-decomposition produced always nonsingular D.

The improvement of β in Step 13 of subalgorithm "verifySparseGen" was called in 61 cases, and the improvement in Step 15 was used in 3 of the 231 tests. With two exceptions β was already improved in line 14 before, so one might skip step 14 and go immediately to step 15. We did not do that because the extended precision calculations in step 15 need significantly more computing time than line 14. The shift s for the Cholesky decomposition in lines 17 – 18 was improved 15 times out of the 211 tests. In all cases the succeeding decomposition did not fail in line 22 and "verifySparseGen" ended successfully. In the median number some 8 power iterations (3.9) were used in line 22. Finally γ was improved 32 times out of the 231 tests in Step 24 of "verifySparseGen", and again improved 2 times in Step 25.

We present some detailed data in Tables 10 - 12. To show all data is too much for this note, so we put the results for all 306 test cases at the `url` in (12.2).

(12.2) `https://www.tuhh.de/ti3/rump/sparselssAllResults.pdf`

Here NaN in the columns for the relative error indicate failure of verification, and otherwise, the columns are self-explaining. The median and maximum relative error of the approximation by `lu` is computed by the error bounds provided by "new". Consequently, there is a "?" for the 5 cases where "new" failed. The ratio of computing times $t_{[57]}/t_{new}$ is only displayed when [57] ended successfully.

In order to reduce space for the results to be displayed in this note, we considered the 20 tests in [57] together with the 306 examples in (12.1) satisfying all properties listed in Table 13. That resulted in 137 test cases filling some 5 pages. Therefore we

TABLE 13
Displayed tests extracted from the 306 *tests in Table 8.*

- `condest(A)` $\leqslant 10^{25}$
- [57] failed with `precond`=1 and was recomputed with `precond`=0
- all tests where "new" failed
- all tests where the median relative error by "new" is larger than 10^{-15}
- all tests where the maximal relative error by "new" is larger than 10^{-10}
- all tests where [57] failed
- all tests where the median relative error by [57] is larger than 10^{-2}
- all tests where the maximal relative error by [57] is larger than 10^{-2}
- all tests where the computing time ratio $t_{[57]}/t_{new}$ is less than 1.84

reduced the number of tests further by moving tests with adjacent numbers in [8] and the same dimension to the `url` in (12.2). Presumably they come from the same source.

That resulted in 84 test cases listed in Tables 10 - 12 filling just 3 pages. That means in particular that if a test is not listed here but only in the `url` in (12.2), then both "new" and [57] succeeded and "new" is at least 1.84 times faster than [57]. The curios ratio 1.84 of computing time $t_{[57]}/t_{new}$ is tuned to fill 3 pages of results. In two cases we observed failure of Matlab's `lu`. In example 1417 from [8] the backslash operator stopped with memory error, and example 1419 caused a crash ending Matlab. That may be due to the limited memory in our laptop.

Numerical evidence suggests that Algorithm `verifySparselss` succeeds to compute verified error bounds for condition numbers close to $\mathbf{u}^{-1}$. The complete list of results in (12.2) shows 5 failures out of the 306 test cases in Table 8, and one of them had an estimated condition number significantly less than 10^{15}. That is for matrix number 934 with `condest(A)`$= 1.7 \cdot 10^{12}$ in [8]. We take a closer look at that case to explain the reason.

For the matrix A of example 934 with dimension $n = 7055$ and $30,082$ nonzero elements we obtain `cond(full(A))`$= 2.5 \cdot 10^{13}$ based on the full singular value decomposition of the sparse matrix. That is a very stable algorithm producing a more reliable estimate, and that is confirmed using the multiple precision package [15]. Moreover, `cond(full(B))`$= 1.2 \cdot 10^{15}$ for the augmented matrix (9.1) shows that there are numerical instabilities because in theory the condition numbers of A and B coincide. And indeed for some right hand sides Matlab's backslash operator produces an approximation with some entries having the wrong sign. Hence, it seems that the problem is more difficult than one might expect by the condition number $= 2.5 \cdot 10^{13}$.

We give some additional test results for randomly generated ill-conditioned sparse matrices using `A = sprand(n,n,dens,1/cnd)` with dimension $n = 10^4$, density 0.001 and `cnd=1e15`. The resulting matrices have some 100,000 nonzero elements each, and the median estimated condition number over the 100 tests was $3.7 \cdot 10^{15}$.

Sometimes generally valid rule of thumbs are only partially satisfied for randomly generated matrices. For example, well conditioned matrix factors are sensitive to perturbations of the input data, while ill-conditioned are not. That is known in the literature [56, 22] but not so much in numerical analysis. It is not clear where this different behaviour stems from; a reason might be that the graph of application matrices is usually structured but that of random matrices is not. Having said this we report the results of our randomly generated tests in Table 14.

TABLE 14
Results for 100 randomly generated ill-conditioned test cases.

	"new"	[57]
inclusions	failed in 3 out of 100 tests	failed in 33 out of 100 tests
median relative error	$3.7 \cdot 10^{-17}$	$1.4 \cdot 10^{-14}$
maximal relative error	$9.4 \cdot 10^{-11}$	761.4
bounds containing 0 in some entries	0 out of 97 successful	24 out of 67 successful

The median condition number $3.6 \cdot 10^{15}$ of our samples is boarder line in the sense that a verification algorithm might just succeed to compute verified bounds. Still, "new" succeeds in 97 cases to compute bounds with at least 10 coinciding figures in each entry. In the median inclusions are close to maximally accurate.

The algorithm in [57] succeeds in 67 out of 100 cases, however, in 24 out of the 67 successful cases some bounds contain zero, i.e., the sign of some entries could not be verified. There was no case were [57] succeeded to compute an inclusion but Algorithm `VerifySparselss` failed.

For the randomly generated ill-conditioned matrices the algorithm in [57] is in the median 0.88 times slower and at most 1.04 times faster than "new". Conversely, "new" is up to 2.9 times faster than [57] and fails in significantly less cases than [57].

In 41 out of the 100 test cases `lu` produced an approximation with some entries having only 1 correct figure, in 6 cases no figure of some entry is correct. In the median "new" is 5.8 times slower than `lu`. The complete set of results can be found at the `url` in (12.2).

We tested Algorithm `verifySparselss` for complex data as well. Some data is shown in the `url` in (12.2). As there were no surprises we refrain from extending our already shown computational data in this note.

We close this note with an example arising from the verification of a three dimensional Navier-Stokes equation using mixed finite elements on a cube domain. The resulting linear system was communicated by Xuefeng Liu [29]. The matrix had dimension $n = 30,424$ with $1,743,841$ nonzero elements. After 18 seconds `verifySparselss` computed an inclusion with median and maximum relative error $4.0 \cdot 10^{-17}$ and $6.0 \cdot 10^{-10}$, respectively. The method in [57] failed, and the built-in "backslash" operator in Matlab delivered an approximation $\tilde{x}$ after 180 seconds. The median relative error of $\tilde{x}$ was $3.6 \cdot 10^{-14}$, however, the relative error also exceeded 1. In fact, for 43 entries the approximate solution by "backslash" had the wrong sign.

13. Conclusion.

We presented Algorithm `verifySparselss` in Table 6 for computing verified error bounds for a linear system with sparse input matrix. The bounds are correct with mathematical certainty including the proof of nonsingularity of the input matrix. The method is applicable to real and complex data including data afflicted with tolerances. Computational evidence suggests that there seems no general purpose method for sparse systems per se as our verification method is sometimes by two orders of magnitude faster than the built-in solver `lu` in Matlab.

The primary goal of our algorithm is to be successful, accepting some penalty in computing time. The second goal is to compute narrow error bounds. In many examples out of the Suite Sparse Matrix Collection [8] our Algorithm `verifySparselss` succeeds to compute accurate error bounds, often with close to maximal accuracy, i.e., all bounds differ by few bits. That applies to randomly generated ill-conditioned sparse systems and a problem related to verification of some Navier-Stokes equation as well.

Acknowledgement. Many thanks to Takeshi Terao and Katsuhisa Ozaki for reading an earlier version of this manuscript and useful comments.

REFERENCES

[1] F.L. Bauer. Optimally scaled matrices. *Numerische Mathematik 5*, pages 73–87, 1963.
[2] B. Breuer, P. J. McKenna, M. Plum. Multiple solutions for a semilinear boundary value problem: a computational multiplicity proof. *J. Differential Equations*, 195:243–269, 2003.
[3] F. Bünger. Verified solutions of two-point boundary value problems for nonlinear oscillators. In *Nonlinear Theory and its Applications (NOLTA), IEICE*, volume E94-N, no.1, 2011.
[4] F. Bünger. Shrink wrapping for Taylor models revisited. *Numerical Algorithms*, 78(4):1001–1017, 2018.
[5] F. Bünger. Preconditioning of Taylor models, implementation and test cases. *Nonlinear Theory and its Applications (NOLTA)*, 12(1):2–40, 2021.
[6] P.A. Businger. Matrices which can be optimally scaled. *Numer. Mathematik*, 12:346–348, 1968.
[7] L. Collatz. Einschließungssatz für die charakteristischen Zahlen von Matrizen. *Math. Z.*, 48:221–226, 1942.

[8] T.A. Davis, Y. Hu: The University of Florida Sparse Matrix Collection. ACM Transactions on Mathematical Software 38, 1, Article 1, 2011.
[9] J.B. Demmel. On floating point errors in Cholesky. LAPACK Working Note 14 CS-89-87, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, 1989.
[10] J. Demmel, Y. Hida. Accurate and efficient floating point summation. *SIAM J. Sci. Comput. (SISC)*, 25:1214–1248, 2003.
[11] I.S. Duff, J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications (SIMAX)*, 22 (4):973–996, 2001.
[12] Iain S. Duff. Ma57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Trans. Math. Softw.*, 30(2):118–144, 2004.
[13] G.E. Forsythe and E.G. Straus. On best conditioned matrices. *Proc. Amer. Math. Soc. 6*, pages 340–355, 1958.
[14] N. J. Higham: *Accuracy and Stability of Numerical Algorithms*, SIAM Publications, Philadelphia, 2nd edition, 2002.
[15] P. Holoborodko. *Multiprecision Computing Toolbox for MATLAB*. Advanpix LLC., Yokohama, Japan, 2019.
[16] R. A. Horn, C. R. Johnson: *Matrix Analysis*, second edition. Cambridge University Press, 2013.
[17] R. A. Horn, C. R. Johnson: *Topics in Matrix Analysis*. Cambridge University Press, 1991.
[18] IEEE Standard for Floating-point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.
[19] R. B. Kearfott, M. T. Nakao, A. Neumaier, S. M. Rump, S. P. Shary, P. van Hentenryck: Standardized notation in interval analysis, *Computational Technologies*, 15(1): 7–13, 2010.
[20] D. E. Knuth: *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison Wesley, Reading, Massachusetts, 1969.
[21] S.P. Kolodziej, M. Aznaveh, M. Bullock, J. David, T.A. Davis, M. Henderson, Y. Hu, R. Sandstrom: The SuiteSparse Matrix Collection Website Interface. Journal of Open Source Software 4, 35, 1244-1248, 2019.
[22] F. R. de Hoog, R. S. Anderssen, M. A. Lukas: Differentiation of matrix functionals using triangular factorization. *Math. Comp.*, 80(275): 1585–1600, 2011.
[23] C.-P. Jeannerod, S.M. Rump. Improved error bounds for inner products in floating-point arithmetic. *SIAM J. Matrix Anal. Appl. (SIMAX)*, 34(2):338–344, 2013.
[24] P. Knight. The Sinkhorn–Knopp algorithm: Convergence and applications. *SIAM Journal on Matrix Analysis and Applications*, 30(1):261–275, 2008.
[25] J. Lahmann, M. Plum. A computer-assisted instability proof for the Orr-Sommerfeld equation with Blasius Profile. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 84(3):188–204, 2004.
[26] M. Lange and S.M. Rump. Error estimates for the summation of real numbers with application to floating-point summation. *BIT*, 57:927–941, 2017.
[27] M. Lange, S.M. Rump. Sharp estimates for perturbation errors in summations. *Math.Comp.*, 88:349–368, 2019.
[28] M. Lange. private communication. 2024.
[29] X. Liu. private communication. 2024.
[30] R. Lohner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen.* PhD thesis, University of Karlsruhe, 1988.
[31] K. Makino and M. Berz. Suppression of the wrapping effect by Taylor model-based verified integrators: long-term stabilization by preconditioning. *International Journal of Differential Equations and Applications*, 10(4):353–384, 2005.
[32] M. Malcolm. On accurate floating-point summation. *Comm. ACM*, 14(11):731–736, 1971.
[33] MATLAB. User's Guide, Version 2023b, the MathWorks Inc., 2023.
[34] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, R. Revol,, S. Torres. *Handbook of Floating-Point Arithmetic.* Birkhäuser Boston, 2nd edition, 2018.
[35] M.R. Nakao. Numerical verification methods for solutions of ordinary and partial differential equations. *Numerical Functional Analysis and Optimization*, 33(3/4):321–356, 2001.
[36] A. Neumaier. Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen. *Zeitschrift für Angew. Math. Mech. (ZAMM)*, 54:39–51, 1974.
[37] A. Neumaier: *Interval methods for systems of equations.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1990.
[38] A. Neumaier. Grand challenges and scientific standards in interval analysis. *Reliable Computing*, 8(4):313–320, 2002.

[39] T. Ogita, S. M. Rump, S. Oishi: Accurate sum and dot product. *SIAM Journal on Scientific Computing (SISC)*, 26(6):1955–1988, 2005.

[40] T. Ogita, S.M. Rump, S. Oishi. Verified Solutions of Sparse Linear Systems by LU factorization, 2005.

[41] S. Oishi, K. Ichihara, M. Kashiwagi, T. Kimura, X. Liu, H. Masai, Y. Morikura, T. Ogita, K. Ozaki, S. M. Rump, K. Sekine, A. Takayasu, N. Yamanaka: *Principle of Verified Numerical Computations.* Corona Publisher, Tokyo, Japan, 2018. [in Japanese].

[42] K. Ozaki, T. Ogita, F. Bünger, S. Oishi. Accelerating interval matri multiplication by mixed precision arithmetic. *Nonlinear Theory and its Applications IEICE*, 6(3):364–376, 2015.

[43] K. Ozaki, T. Ogita, S. Oishi: Error-free transformation of matrix multiplication with a posteriori validation. *Numer. Linear Alg. Appl.*, 23(5):931–946, 2016.

[44] S.M. Rump. *Kleine Fehlerschranken bei Matrixproblemen.* PhD thesis, Universität Karlsruhe, 1980.

[45] S.M. Rump. Validated Solution of Large Linear Systems. In R. Albrecht, G. Alefeld, H.J. Stetter, editors, *Validation numerics: theory and applications*, volume 9 of *Computing Supplementum*, pages 191–212. Springer, 1993.

[46] S.M. Rump. Verified Computation of the Solution of Large Sparse Linear Systems. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 75:S439–S442, 1995.

[47] S. M. Rump: INTLAB – INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Springer Netherlands, Dordrecht, 1999.

[48] S.M. Rump. Verified Solution of Large Linear and Nonlinear Systems. In H. Bulgak, C. Zenger, editors, *Error Control and adaptivity in Scientific Computing*, pages 279–298. Kluwer Academic Publishers, 1999.

[49] S. M. Rump: Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010.

[50] S.M. Rump. Fast Interval Matrix Multiplication. *Numerical Algorithms*, 61(1):1–34, 2012.

[51] S.M. Rump, C.-P. Jeannerod. Improved backward error bounds for LU and Cholesky factorizations. *SIAM J. Matrix Anal. Appl. (SIMAX)*, 35(2):684–698, 2014.

[52] S.M. Rump, M. Lange. Fast computation of error bounds for all eigenpairs of a Hermitian and all singular pairs of a rectangular matrix with emphasis on eigen- and singular value clusters. *Journal of Computational and Applied Mathematics (JCAM)*, 434, 2023.

[53] S.M. Rump, T. Ogita. Super-fast validated solution of linear systems. *Journal of Computational and Applied Mathematics (JCAM)*, 199(2):199–206, 2006. Special issue on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN 2004).

[54] A. van der Sluis. Condition numbers and equilibration of matrices. *Numer. Mathematik*, 14:14–23, 1969.

[55] H.J. Stetter, Sequential defect correction in high-accuracy floating-point arithmetics. In: Griffiths, D.F. (eds) Numerical Analysis, Lecture Notes in Math., 1066, pp. 186–202, 1984.

[56] G. W. Stewart: On the perturbation of LU, Cholesky, and QR factorizations. *SIAM J. Matrix Anal. Appl.*, 14: 1141–1146, 1993.

[57] Terao T., K. Ozaki. Method for verifying solutions of sparse linear systems with general coefficients. 2024. `https://arxiv.org/abs/2406.02033`.

[58] J.H. Wilkinson. *The algebraic eigenvalue problem.* Clarendon Press, Oxford, 1965.

[59] T. Yamamoto. Error Bounds for Approximate Solutions of Systems of Equations. *Japan J. Appl. Math.*, 1:157–171, 1984.

[60] Y.-K. Zhu, J.-H. Yong, G.-Q. Zheng. A new distillation algorithm for floating-point summation. *SIAM J. Sci. Comput.*, 26(6):2066–2078, 2005.

[61] G. Zielke, V. Drygalla. Genaue Lösung linearer Gleichungssysteme. *GAMM Mitt. Ges. Angew. Math. Mech.*, 26:7–108, 2003.