To appear in SIMAX, 2026

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 are included in INTLAB, the Matlab/Octave toolbox for reliable computing. Despite several efforts, the verified solution of general sparse linear systems was 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 to compute entrywise error bounds for the solution of general real or complex sparse systems 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 $A \approx L_1L_2$ 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 maximally accurate, i.e., the bounds differ by few bits.

We present a new a priori error bound based on Perron-Frobenius Theory on the residual of a floating-point Cholesky decomposition which improves on existing ones by some two orders of magnitude. Our three subalgorithms benefit from this new bound.

Based on the results in this Part I, an alternative approach will be presented in Part II of this note. Those methods are somewhat simpler, but often slower. However, they seem to be more robust, i.e., 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. Moreover, 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 [35], 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.

*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).

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 cf. [39, 52, 42] and the literature cited over there. Many verification algorithms are included in INTLAB [49], 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 [32, 37, 3, 28, 33, 4, 5, 6], however, it seems difficult to provide general purpose verification algorithms.

For given symmetric (positive definite) A it is proposed in [47] 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. [56]) that method used a priori estimates on $\|E\|_2$ based on Demmel's result [11], see also [17, 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 $\mathrm{cond}(A) \lesssim 10^8$ in double precision (binary64). That is the reason why [52, Challenge 10.15] asks for a verification method for sparse linear systems of reasonable size with $\mathrm{cond}(A) \geqslant 10^{10}$.

Most methods to solve full linear systems use an approximate inverse as preconditioner which is prohibitive for sparse system matrix. The method [41] 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 [47], 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

$$\sigma_{\min}(A) \geqslant \sigma_{\min}(\tilde{L}_1)\sigma_{\min}(\tilde{L}_2) - \|A - \tilde{L}_1\tilde{L}_2^T\|_2 \tag{1.2}$$

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 $\mathrm{cond}(A)^{1/2}$, then $\mathrm{cond}(F^TF) \approx \mathrm{cond}(A)$ and those methods work fine. However, not too many details were given in [47].

Next we proved the following theorem [48, 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*

$$\sigma_{\min}(A) > \tilde{\lambda} - \max\{\|A - \tilde{\lambda}I - \tilde{L}_1\tilde{D}_1\tilde{L}_1^T\|, \|A + \tilde{\lambda}I - \tilde{L}_2\tilde{D}_2\tilde{L}_2^T\|\}. \tag{1.3}$$

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

$$\sigma_{\min}(A) > \tilde{\lambda} - \|A - \tilde{\lambda}I - \tilde{L}_1\tilde{D}_1\tilde{L}_1^T\|. \tag{1.4}$$

This approach needs two LDL^T-decompositions and is applicable for condition numbers of A close to $\mathbf{u}^{-1} \approx 10^{16}$. In [50] 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 [47, 48, 50] were not further pursued because the LDL^T-decomposition was not robust enough to allow for a verification method satisfying my expectations.

Nowadays good scaling and equilibration routines are available [13, 14] making those methods attractive. That was observed by Terao and Ozaki [61] and triggered our note in two parts. They also 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[1] of B is known to be $(n, n, 0)$, the lower bound on $\sigma_{\min}(A) = \sigma_{\min}(B) \geqslant s - \|B - \tilde{L}\tilde{D}\tilde{L}^T\|_2$ follows if the inertia of $\tilde{D}$ is $(n, n, 0)$ as well. Compared to Theorem 1.1 only one LDL^T-decomposition is necessary, however, of a matrix of double size. They use in particular the preconditioning in [13] to ensure robustness 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 [56] that nevertheless a residual iteration based on $\tilde{L}, \tilde{D}$ works, and that is used by Terao and Ozaki [61]. We come to that again in Section 11.

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 in [56] for (1.1) by 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 $\mathrm{cond}(F_1) \approx \mathrm{cond}(A)^{1/2}$. 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 [34, 38, 12, 40, 65, 64]. In contrast, an inclusion of $A - LDL^T$ is computed in two steps with an interval factor in the second product causing additional overestimation.

In the three cases the lower bound on the singular value is eventually computed by the successful termination of a floating-point Cholesky decomposition $A \approx \tilde{R}^T\tilde{R}$. To that end an a priori upper bound on the residual $\|A - \tilde{R}^T\tilde{R}\|$ is used, and, if not sufficiently good, improved by more time consuming methods. Our new a priori bound based on Perron-Frobenius Theory comes at very low cost and improves in the median on existing bounds by two orders of magnitude, cf. Figure 2.

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 [10] and found linear systems where our verification method is by two orders of

[1] Number of positive, negative and zero eigenvalues.

magnitude faster than the built-in backslash Matlab operator (but also vice versa). That should not happen because our verification methods include the computation of an approximate solution of the linear system.

As test matrices we took all real square matrices of the Suite Sparse Matrix Collection [10] with dimension n satisfying $10^4 \leqslant n \leqslant 10^6$, condition number $\kappa_2(A)$ with $10^7 \leqslant \kappa_2 \leqslant 10^{16}$ and not exceeding one million entries, plus all test cases in [61]. That resulted in 400 real test cases.[2] In 399 cases we could compute verified and accurate 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. The methods presented in Part II of this note failed in none of the examples including complex, rectangular and random linear systems.

Our primary target is that our algorithm ends successfully, i.e., verifies nonsingularity of the input matrix and computes rigorous error bounds for the solution of the linear system. Our algorithms are 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 many cases with maximum relative error $\lesssim 10^{-15}$, i.e., close to maximally accurate bounds in double precision (binary64). 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 [61].

Our notation is as follows. We assume a set of floating-point numbers $\mathbb{F}$ with an arithmetic according to the IEEE 754 floating-point standard [21] to be given. We use double precision (binary64) in a nearest rounding[3] 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 $\mathrm{float}(\cdot)$ 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 adaptation of the following to intermediate results outside the floating-point range is rather straightforward. For simplicity, we assume throughout this note that neither overflow nor underflow occurs.

The error of a single operation $\circ \in \{+, -, \times, /\}$ of floating-point numbers a, b is bounded by [17, Theorems 2.2 and 2.3]

$$|\mathrm{float}(a \circ b) - a \circ b| \leqslant \mathbf{u} \cdot \min\left(|a \circ b| \ , \ |\mathrm{float}(a \circ b)| \right). \tag{1.5}$$

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, (1.5) remains true when replacing $\mathbf{u}$ by $2\mathbf{u}$. The Matlab/INTLAB command `setround(rnd)` switches the rounding mode in the sense that, for example, `rnd` = 1 implies that henceforth all operations are executed in rounding upwards until the next call of `setround`. Similarly, `rnd` = −1 switches the rounding downwards and `rnd` = 0 to nearest.

Our goal is to calculate mathematically correct but also accurate inclusions for the

[2] In addition, there are three complex test cases satisfying the conditions which are treated in Part II of this note because [61] handles only real matrices.

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

solution of a sparse linear system $Ax = b$. To that end we use the following notations:

$$
\begin{array}{ll}
[\![expr]\!]_{2,1} & \text{evaluation in extended precision, result rounded into } \mathbb{F} \\
\langle expr \rangle & \text{inclusion in double precision using directed roudings in } \mathbb{F} \\
\langle\!\langle expr \rangle\!\rangle_{2,1} & \text{inclusion computed in extended precision and rounded into } \mathbb{F}
\end{array} \tag{1.6}
$$

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 (1.6) are used exclusively for expressions where each entry is computable by a dot product. Typical examples for $[\![\cdot]\!]_{2,1}$ are $b - Ax$ or $A - R^T R$, and the evaluation is usually but not necessarily in a nearest rounding.

For the two latter notations for inclusions in (1.6) the expression has to satisfy an additional property: When computing the expression in rounding downwards, the computed result must be a mathematically correct lower bound of the true result, and similarly for rounding upwards. Consider the following code

```
setround(-1); P = A*B; Q = C*D; S = P + Q;
```

for compatible factors A, B and C, D. Then each entry of P, Q and S is a lower bound of the corresponding entry of AB, CD and $AB + CD$, respectively. That assertion holds similarly for rounding upwards, and as a consequence AB or $AB + CD$ are suitable expressions for $\langle\cdot\rangle$ and $\langle\!\langle\cdot\rangle\!\rangle_{2,1}$. However, $AB - CD$ is not suitable because $P - Q$ is not necessarily a lower bound of $AB - CD$. Similarly $b - Ax$ and $A - R^T R$ are not suitable, but $Ax - b$ and $R^T R - A$ are.

For the implementation of $[\![\cdot]\!]_{2,1}$ and $\langle\!\langle\cdot\rangle\!\rangle_{2,1}$ any of the many accurate dot product algorithms may be used. Special care is necessary for $\langle\!\langle\cdot\rangle\!\rangle_{2,1}$ because mixed precisions and directed rounding are used. It is mandatory that the arithmetic operations and the rounding into $\mathbb{F}$ respect the rounding mode.

In [61] the toolbox Advanpix [18] was used, a very fast multiple-precision Matlab package emulating a large number of Matlab's algorithms. In order to have a fair comparison with [61] we used Advanpix in this note as well.[4] 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} satisfying the IEEE 754 floating-point standard [21]. We use this precision throughout this note. Sample executable Matlab/INTLAB codes for the expressions in (1.6) for $Ax - b$ are

```
[[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);
```

(1.7)

First note that the type cast `mp(x)` is error-free, i.e., `x = mp(x)`. Second, the type cast ensures that `A*mp(x)` is computed in extended precision with extended precision

[4] There is a fast Matlab implementation of a new method which is included in INTLAB [49] and will be used in Part II of this note. Since [61] uses the Advanpix toolbox [18], we use it in the present Part I as well for a fair comparison.

result, and in turn that ensures that the difference in `A*mp(x)-b` is computed in extended precision as well. Moreover, the type cast `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.

Corresponding to the lower bound for the smallest singular value of A we often need, as in (1.1) or (1.2), upper bounds for the spectral norm of a matrix. It is common to use $\|P\|_2 \leqslant \sqrt{\|P\|_1\|P\|_\infty}$, however, it is too weak for our purposes. Perron-Frobenius Theory and [9] imply for square P and any positive vector x the better bound

$$\|P\|_2 \leqslant \||P|\|_2 = \sigma_{\max}(|P|) = \sqrt{\lambda_{\max}(|P|^T|P|)} \leqslant \max_k \frac{\big(|P|^T(|P|x)\big)_k}{x_k} \tag{1.8}$$

for general P, and

$$\|P\|_2 \leqslant \max_k \frac{(|P|x)_k}{x_k} \tag{1.9}$$

for symmetric/Hermitian P. To that end we developed in [55] the following Algorithm `NormBnd` in Table 1 computing a rigorous upper bound `N` of $\||A|\|_2$. For symmetric and/or Hermitian A the parameter `herm` can be set to *true*, otherwise it must be *false*.

```
function N = NormBnd(A,herm)
  setround(0), x = ones(size(A,1),1); M = [1 2]; iter = 0;
  A = mag(A);   % matrix of entrywise absolute values
  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
```

TABLE 1
Algorithm to compute a rigorous upper bound of $\||A|\|_2$*; herm=true if, and only if,* $A^H = A$

That algorithm is used in [61] as well. Numerical evidence suggests that `NormBnd` improves $\sqrt{\|A\|_1\|A\|_\infty}$, an upper bound on $\||A|\|_2$ as well, often by a factor 2 and more. In our applications `NormBnd` is used for sparse matrix residuals and seems to overestimate $\|A\|_2$ not too much. Besides, we do not know of a good upper bound for $\|A\|_2$ which can be computed within reasonable time.

We use standard singular value perturbation bounds [20, Theorem 3.3.16] for real or complex $n \times n$ matrices A, E, namely

$$|\sigma_k(A + E) - \sigma_k(A)| \leqslant \|E\|_2 \tag{1.10}$$

for $\sigma_1 \geqslant \ldots \geqslant \sigma_n$ denoting the singular values and $k \in \{1, \ldots, n\}$, and the eigenvalue perturbation bound [19, (6.3.4.1)]

$$\lambda_k(A) + \lambda_{\min}(E) \leqslant \lambda_k(A+E) \leqslant \lambda_k(A) + \lambda_{\max}(E) \tag{1.11}$$

for real or complex Hermitian $n \times n$ matrices A, E and $k \in \{1, \ldots, n\}$. Moreover, for $A, B \in \mathbb{R}^{n\times n}$ we use

$$\sigma_{\min}(AB) \geqslant \sigma_{\min}(A)\sigma_{\min}(B) \tag{1.12}$$

which follows by $\dfrac{\|ABx\|_2}{\|x\|_2} \geqslant \dfrac{\|ABx\|_2}{\|Bx\|_2/\sigma_{\min}(B)} \geqslant \sigma_{\min}(A)\sigma_{\min}(B)$. A real or complex signature matrix S is diagonal with $|S_{kk}| = 1$ for all k. For vectors (and similarly for matrices) we use $|\cdot|$ to denote entrywise absolute values, and use $x \leqslant y$ to denote 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 commonly used $\gamma_k := \frac{k\mathbf{u}}{1-k\mathbf{u}}$, cf. [17]. 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[5] 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 symmetric 2×2 matrices. We show how to handle complex linear systems, data afflicted with tolerances, and present[6] Algorithm `verifySparselss1` 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 [61] 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 IEEE 754 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.

Our target is to estimate a residual $\|AB-C\|_2$ for compatible large sparse matrices A, B, C. We may use $\langle AB - C\rangle$ as in (1.7), but that requires 2 matrix products per se. Following we describe a 2-step approach computing $AB - C$ in rounding upwards together with a cheap but weaker a priori error estimate. Only if not sufficient, $AB-C$ is computed in rounding downwards as well. To that end we need a computable rigorous error estimate for $AB - C$ computed with faithful rounding.

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

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

[5] In fact, for nearly symmetric/Hermitian matrices as will be shown in Part II.

[6] Another Algorithm `VerifySparselss2` is presented in Part II of this note.

in a nearest rounding was shown in [26]. 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 $\mathrm{float}(AB)$ of AB for $A \in \mathbb{F}^{m\times\ell}, B \in \mathbb{F}^{\ell\times n}$ is bounded by

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

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. However, we need a new error estimate like (2.2) for directed rounding because the methods in [26] cannot be used.

The first bound for directed rounding was given by Ozaki [43], namely $|\mathrm{float}(AB) - AB| \leqslant 2(\mu+4)\mathbf{u}AB$. It was designed for mixed-precision calculations. The bound requires $4\mu \leqslant \mathbf{u}^{-1}$ but also that both A, B are nonnegative. For general A, B it was shown in [30, Corollary 2.4] that

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

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

The assumption $\mu \leqslant (2\mathbf{u})^{-1/2}-1$ bounding the number of nonzero products seems hardly an obstacle when using double precision (binary64), i.e. $\mu \leqslant 67,108,863$ nonzero products per entry. However, recently often mixed-precision is used [8] in the sense that a linear system is solved in lower precision and the solution improved by a residual iteration in higher precision. That approach saves time and memory, and it is even used for matrices with condition number much larger than the common bound $\mathbf{u}^{-1}$. That observation goes back to our internal report [46] from 1990 which had to wait to be put into a journal paper [51] until 2009 when we could analyse the behaviour. To make the point, we constructed for the latter paper an extreme example of a floating-point matrix with true condition number $1.1 \cdot 10^{305}$ treated in double precision. The important key is the availability of high-precision dot products.

Having said this the assumption $\mu \leqslant (2\mathbf{u})^{-1/2}-1$ bounding the number of nonzero products in single precision to $\mu \leqslant 2,895$ is restrictive. Therefore we use the following Lemma 2.1 suitable up to $\mu \leqslant 4.5 \cdot 10^{15}$ nonzero products per entry in double, and up to $\mu \leqslant 8,388,609$ in single precision.

LEMMA 2.1. *Let $A \in \mathbb{F}^{m\times\ell}$ and $B \in \mathbb{F}^{\ell\times n}$ be given, and let $\mathit{float}(AB)$ be calculated in a faithful rounding. Denote by μ the maximum number of nonzero products to compute the entries of AB. If $4(\mu-1)\mathbf{u} \leqslant 1$, then*[7]

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

Proof. Let $z \in \mathbb{F}^\ell$ be a vector of floating-point numbers, and let $\mathrm{float}(\sum_{k=1}^{\ell} z_k)$ be computed in some faithful rounding in any order. Then [29, Corollary 3.3] shows

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

provided that the vector z has not more than μ nonzero elements and $\mu \leqslant 1 + \mathbf{u}^{-1}/2$.

Let $x, y \in \mathbb{F}^\ell$ be given and denote $z_k := \mathrm{float}(x_k y_k)$ for $k \in \{1, \ldots, \ell\}$ so that

[7] It will be clear from the proof that with the weaker assumption $2(\mu-1)\mathbf{u} \leqslant 1$ the assertion (2.4) remains true when replacing $(2\mu+1)$ by $(2\mu+2)$.

$\mathrm{float}(x^T y) = \mathrm{float}(\sum_{k=1}^{\ell} z_k)$ with operations computed in some faithful rounding. Then

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

by (1.5). Hence the assumption $4(\mu-1)\mathbf{u} \leqslant 1$ implies

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

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

Remark 2.2. For a bound of type (2.4) it is mandatory to limit the number of nonzero products μ. Consider $x^T y$ with x denoting the vector of $1's$ and $y = (1, e, e, \ldots, e)^T \in \mathbb{R}^{\ell}$ for a tiny number e and $\ell = \mathbf{u}^{-1}/2 + 1 =: \mu$. In rounding upwards the result of $1+e$ is the successor $1+2\mathbf{u}$ of 1, so that the floating-point result of $x^T y$ is $1 + 2(\ell-1)\mathbf{u} = 2$. The error for tiny e is $2 - (1 + (\ell-1)e) \approx 1 = 2(\mu-1)\mathbf{u}$ as in (2.5). However, when adding one more e in rounding upwards, the result is the successor of 2, i.e., $2 + 4\mathbf{u}$ with an additional error $4\mathbf{u}$ and (2.5) would fail.

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 2.3. *Let $A \in \mathbb{F}^{n\times\ell}$ 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 or ℓ. For $C \in \mathbb{F}^{n\times n}$ and $E := \mathit{float}(AB - C)$ it follows

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

without limit on n or ℓ. Denote by $\overline{\mu}$ the maximum number of nonzero products in the products $(AB)_{ij}$. If a faithful rounding is used and $\overline{\mu} \leqslant (2\mathbf{u})^{-1/2} - 1$, then (2.6) *and* (2.7) *remain true when replacing $\mathbf{u}$ by $2\mathbf{u}$. For faithful rounding and $4(\overline{\mu}-1)\mathbf{u} \leqslant 1$,* (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

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

First, let $\widehat{\varrho}$ and σ denote the column vectors with elements $\mu_i\varrho_i$ and σ_j, respectively. Then using $\min(\mu_i, \nu_j) \leqslant \mu_i$ and the outer product $\widehat{\rho}\sigma^T$ it follows

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

Second and similarly, denoting by $\widehat{\sigma}$ the column vector with elements $\nu_j\sigma_j$ gives

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

and implies (2.6). Using $P := \mathrm{float}(AB)$ and (1.5) gives

$$\begin{aligned}\|\mathrm{float}(AB-C) - (AB-C)\|_2 &= \|\mathrm{float}(P-C) - (P-C) + (P-AB)\|_2 \\ &\leq \mathbf{u}\|E\|_2 + \|P-AB\|_2\ ,\end{aligned}$$

and (2.6) proves (2.7). For faithful rounding the estimates follow by (2.3) and (2.4).∎

Corollary 2.4. *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*

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

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

$$\|\mathit{float}(AA^T) - AA^T\|_2 \leq 2\mathbf{u}\sum_{k=1}^{n}\mu_k\left(\mathit{float}(AA^T)\right)_{kk}. \tag{2.9}$$

If $4(\max\mu_k - 1)\mathbf{u} \leq 1$, 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 2.3 implies in a nearest rounding

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

In rounding upwards $(AA^T)_{kk} \leq \left(\mathrm{float}(AA^T)\right)_{kk}$ 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 $\|AB - C\|_2$. We do that in three stages. First, we compute $AB - C$ in rounding upwards and use the a priori estimate in (2.7). If not successful, then we compute $AB - C$ in rounding downwards to obtain an inclusion of $AB - C$. If still not successful, accurate dot products $\langle\!\langle AB - C\rangle\!\rangle_{2,1}$ are used as in (1.7).

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 $\max\mu_k \leq (2\mathbf{u})^{-1/2} - 1 = 67,108,863$. If only $\max\mu_k \leq \mathbf{u}^{-1}/4 \approx 2.3\cdot 10^{15}$ is satisfied, then the code is adapted according to Lemma 2.3.

Lemma 2.5. *Let $A \in \mathbb{F}^{n\times\ell}$, $B \in \mathbb{F}^{\ell\times n}$ and $C \in \mathbb{F}^{n\times n}$. Executing the Matlab code*

```
setround(1); E = A * B - C; u2 = pow2(-52);
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 = (1 + u2) * NormBnd(E,false) + u2 * errAB;
```
(2.10)

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

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

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

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

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

```
mp.Digits(34); v = pow2(-113);
setround(0); G = abs(double(C - mp(A) * B));
setround(1);
normG2 = NormBnd(G + realmin * spones(G), false);
errAtA = sum(spones(A), 2)' * sqr(vecnorm(A, 2, 2));
alpha = normG2 + v * (normG2 + errAtA);
```
(2.13)

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

Remark 2.6. Note that the codes in (2.10) and (2.12) produce column vectors `mu` and `rho`, and row vectors `nu` and `sigma`. Hence `errAB` $\in \mathbb{F}$.

Remark 2.7. In order to compute mathematically correct bounds directed roundings are used. Moreover, in the calls of `NormBnd` from Table 1 the second parameter can be replaced by `true` for Hermitian input.

Remark 2.8. The codes in (2.12) and (2.13) avoid to store an extended precision matrix.

Remark 2.9. 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 2.3. Define $P := \mathrm{float}(AB)$. Then $\mathrm{float}(1 + \mathbf{u}2) = 1 + 2\mathbf{u}$ is the successor of 1, such that rounding upwards and (1.5) imply $|P - C| \leqslant (1 + 2\mathbf{u})|\mathrm{float}(P - C)|$. Hence,

$$\|P - C\|_2 \leqslant \| \, |P - C| \, \|_2 \leqslant (1 + 2\mathbf{u}) \| \, |\mathrm{float}(P - C)| \, \|_2 = (1 + 2\mathbf{u}) \| \, |E| \, \|_2,$$

and (2.7) proves $\|C-AB\|_2 \leqslant \alpha$ because `NormBnd` yields an upper bound on the 2-norm of the absolute value of E. To show (2.11) define

$$\texttt{setround}(-1);\ \texttt{E1} = \texttt{A} * \texttt{B} - \texttt{C};$$
$$\texttt{setround}(+1);\ \texttt{E2} = \texttt{A} * \texttt{B} - \texttt{C};$$

Note that `E2` is the matrix `E` in (2.10) and `E1` is implicitly computed in (2.11). Then the rounding modes imply[8] $\texttt{E1} \leqslant AB - C \leqslant \texttt{E2}$ with entrywise comparison. Hence $|AB - C| \leqslant \max(|\texttt{E1}|, |\texttt{E2}|)$ and $\|C - AB\|_2 \leqslant \beta$ follows.

The third code (2.12) uses extended precision in the mp-toolbox [18]. In rounding to nearest `double(abs(M)) = abs(double(M))` for a given mp-matrix `M` because taking absolute values does not create rounding errors, neither in double nor in extended precision, and the type cast `double` from mp to double respects the rounding mode to nearest. Therefore, the second line in (2.12) is equivalent to computing `F := abs(C-mp(A)*B)` in extended precision and rounding to nearest with relative rounding error $\mathbf{v} := 2^{-113}$, followed by `G = double(F)`. Hence, the entries $\texttt{G}_{ij} \in \mathbb{F}$ are equal to $|\texttt{F}_{ij}|$ or one of the immediate floating-point neighbours of the extended precision numbers $|\texttt{F}_{ij}|$, and in rounding upwards[9] $\texttt{F}_{ij} \leqslant \text{float}(\texttt{G}_{ij} + \texttt{realmin}) \in \mathbb{F}$ and $\|\texttt{F}\|_2 = \||\texttt{C} - \texttt{mp(A)} * \texttt{B}|\|_2 \leqslant \texttt{normG2}$.

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 2.3. Denote $\texttt{M} := \texttt{mp(A)} * \texttt{B}$. Then $|C - M| \leqslant (1 + \mathbf{v})\text{float}_{mp}(|C - M|) = (1 + \mathbf{v})\texttt{F}$ by (1.5), and (2.6) implies[10]

$$\|C - AB\|_2 = \|C - \texttt{M} + \texttt{M} - AB\|_2 \leqslant (1 + \mathbf{v})\texttt{normG2} + \mathbf{v} \sum_{k=1}^{n} \min(\mu_k, \nu_k)\rho_k\sigma_k \ . \tag{2.14}$$

The fourth code (2.13) uses again the mp-toolbox [18]. By assumption the sets 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

$$\texttt{mu} = \texttt{sum}(\texttt{spones}(\texttt{A}), 2) = \texttt{sum}(\texttt{spones}(\texttt{B})) = \texttt{nu}'$$

and

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

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 $\|C - AB\|_2 \leqslant \| |C - AB| \|_2$, which in turn is small in our applications.

For the special case of Cholesky decomposition $A \approx \tilde{R}^T\tilde{R}$ there is an a priori estimate [56, Lemma 2.2], [17, Theorem 10.5] of the residual $\|\tilde{R}^T\tilde{R} - A\|_2$ using only the fact that a Cholesky decomposition ran to completion, not using the factor $\tilde{R}$. We improve this estimate by applying Perron-Frobenius Theory.

LEMMA 2.10. *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*

[8] Note that this is true for using `A*B-C`, but would not necessarily be true when using `C-A*B`.
[9] Note that `G+realmin` instead of `G+realmin*spones(G)` would produce a full matrix.
[10] It is better to use `gamma = normG2 + v*(normG2 + errAB)` to keep small terms together.

$\Phi_{ij} := \min(\mu_i, \mu_j) + 1$ *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 $4(\max\mu_k - 1)\mathbf{u} \leqslant 1$, *then the estimate remains true when using* $\Phi_{ij} := \min(\mu_i, \mu_j) + 2$ *and replacing* $\mathbf{u}$ *by* $2\mathbf{u}$.

Remark 2.11. 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 2.12 to compute an improved upper bound for $\|\Delta A\|_2$ efficiently.

Proof. In [54] 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 $i = j$. Carefully going through the proof of Theorem 4.4 in [54] 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 [17, 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 and positive, 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 [9], [19, 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 [31] who provided the ingenious piece of Matlab code in (2.18).

Corollary 2.12. *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*

```
rcx = cumsum(x,1,'reverse');
vx = v.*x;
w = cumsum(vx) - vx + v.*rcx;
```
(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 by the definition of Φ.

The previous estimate [56, Lemma 2.2], [17, Theorem 10.5] continues from (2.16) by replacing the entries φ_{ij} of Φ in (2.15) by $\sqrt{\varphi_{ii}\varphi_{jj}}$, where $\varphi_{kk} := \gamma_{k+1} = \frac{(k+1)\mathbf{u}}{1-(k+1)\mathbf{u}}$. 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

$$\|\Delta A\|_2 \leqslant \frac{\gamma_{k+1}\mathbf{u}}{1-\gamma_{k+1}\mathbf{u}}\ \mathrm{trace}(A). \tag{2.19}$$

Executing the code in (2.18) in rounding upwards computes an upper bound for Φx because the quantities involved are positive. We show in Figure 2 in Section 12 numerical evidence that the new estimate (2.15) together with Corollary 2.12 improves upon (2.19) in the median by two orders of magnitude.

3. Scaling, equilibration and approximation of smallest singular value. Our verification methods require 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 $\kappa_2(A)$ 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 [7, Lemma 1] is symmetric

$$\inf_{D_1, D_2 \in \mathcal{D}_n} \kappa_2(D_1 A D_2) = \inf_{D\in\mathcal{D}_n} \kappa_2(DAD)\ .$$

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 [58, Theorem 4.3]

$$\kappa_2(A) \leqslant q \min_{D\in\mathcal{D}_n} \kappa_2(D^H A D)$$

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[11] 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 [16, 1]. To that end the famous Sinkhorn-Knopp algorithm is the algorithm of choice. For a good introduction and historical remarks see [27]. 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 a good starting vector for symmetric positive definite A is

[11] This includes multiple right hand sides $b \in \mathbb{R}^{n\times k}$.

`1./sqrt(diag(A))`. The following iteration scales columns and rows alternately:

```
d = 1./sqrt(diagA);
for k = 1:4, d = 1./(abs(A) * d); end
A = (d. * A). * d';
```
(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.,

```
d = 1./vecnorm(A,2)';
for k = 1:4, d = 1./(abs(A) * d); end
A = (d. * A). * d';
```
(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 on rows and columns [27]:

```
[p,row,col] = equilibrate(A,'vector');
for k = 1:2
   col = 1./(abs(A(p,:)') * row);  row = 1./(abs(A(p,:)) * col);
end
row = sign(row). * pow2(round(log2(abs(row))));
col = sign(col). * pow2(round(log2(abs(col))));
A = row. * A(p,:). * col';
```
(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 in magnitude 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.

The main purpose of equilibration is to reduce the condition number of a matrix. For our equilibration as in (3.1), (3.2) and (3.3) there are counterexamples where the condition number increases after equilibration, but they are rare. Numerical evidence to that is postponed to Part II of this note (cf. Section 10, Figure 2) because we will compare not only the condition numbers with and without equilibration but also the behaviour of our algorithms in both parts of the note.

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

```
[R,FLAG,p] = chol(A,'vector');
```
(3.4)

producing an error flag and permutation information. If successful, i.e., `Flag` = 0, it follows $R^T R \approx P^T A P$ for the permutation matrix P mapping $\{1,\ldots,n\}$ into `p`. 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" [14]. 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 robust result. The maximum threshold is `thresh = 0.5`, and we always use this value.

In previous Matlab releases up to and including R2025b there is an obstacle when applying `ldl` to an augmented matrix $B := \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$. Mathematically, the factor D of B consists only of 2×2 blocks, each 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 happened that (3.5) computed 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 about a quarter of our test cases. In such a case the part of L corresponding to zero blocks in D are the rows of the identity matrix. Therefore, 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.

Let a symmetric $n \times n$ matrix M be given. Another remedy to avoid that the factor D of Matlab's `ldl` applied to M becomes singular is to use

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

when $nnz(D) < n$, i.e., the original factor D is singular. 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}:

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

In our application it is safe to use the absolute shift by 10^{-50} because the input matrices are scaled to have 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` until release 2025b. The reason is that $MA57$ [14] uses a "zero pivot tolerance" 10^{-20}. Unfortunately, that is applied in the Matlab implementation not only to L but also to D, eventually producing a singular factor D. When changing the tolerance to zero, no singular factor D appears any more. However, in Matlab the user cannot change that tolerance. After reporting that behaviour to MathWorks the bug was removed, and from Version 2026a of Matlab a singular factor D is no longer produced.

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 often fast, however, also sometimes very slow and inaccurate.

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

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

That seems a robust 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 $\tilde{s}(L) \lesssim \sigma_{\min}(A)$ by

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

Note that only L is needed for the iteration. 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 matrix 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 [63]. To further improve the accuracy we store an approximate solution as a pair $(\tilde{x}, \tilde{y})$ interpreted as an unevaluated sum $\tilde{x} + \tilde{y}$. This technique was introduced in [45] and later called "staggered correction" [59]. Together with accurate dot products it often allows for almost maximally accurate error bounds.

We sketch in Table 2 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 Line 3 is calculated in extended precision, the unevaluated sum $\tilde{x} + \tilde{y}$ should be a good approximation to $A^{-1}b$. The fourth line[12] 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 small, correcting less significant bits of $\tilde{y}$. That is utilized in Line 6. When computing

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

[12]The call `[x,y] = TwoSum(a,b)` computes $x = \mathrm{float}(a + b)$ for scalars, vectors and matrices a, b, and in addition y such that $x + y = a + b$ is mathematically correct [36].

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

TABLE 2
Residual iteration and inclusion of the solution $A^{-1}b$ using a rigorous lower bound $0 < s \leqslant \sigma_{\min}(A)$

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

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

The function `vecnorm` in Line 9 denotes $\|\varrho\|_2$ for a column vector ϱ, and `vecnorm(M)` is the row vector of Euclidean norms of the columns of a matrix `M`. First, suppose b is a vector. Then proceeding as in [56] and abbreviating the vector of all $1's$ by $\mathbf{e}$ we obtain

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

because $s \leqslant \sigma_{\min}(A)$ and the computation of δ in the last line is in rounding upwards. For multiple right hand sides $b \in \mathbb{R}^{n,k}$ apply (4.1) successively to the columns of b.

The residuals are computed using the extended precision package Advanpix in [18] 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. In this Part I we use Advanpix for a fair comparison to [61], in Part II we show how to use higher precision to achieve almost always maximally accurate inclusions for all entries of the solution.

Using accurate dot products is mandatory and ensures to obtain accurate entry-wise error estimates. To see that we display in Table 3 the intermediate results for the residual iteration in Table 2 for two representative examples. The examples are number 1210 and 438 of [10], the first one being symmetric, the second one general. As we will see later the algorithm in [61] could not compute verified bounds for both examples, in the first example due to the large condition number, and in the second example, although the condition number is only $2.1 \cdot 10^{10}$, the algorithm failed to compute a positive lower bound for the smallest singular value, both with and without preconditioning.

The input is normed to $\|A\|_\infty = 1$ and the right hand side is computed such that the solution $A^{-1}b$ is approximately the vector of 1's. The smallest singular value in Line 4 of Table 3 shows that in particular the first matrix is ill-conditioned because, since A is normed to 1, the reciprocal of $\sigma_{\min}(A)$ is the condition number. The $\tilde{x}$ in Line 5 is computed by $A\backslash b$ followed by one residual iteration in working precision to ensure backward stability, cf. [57]. It is a well known fact in numerical analysis that, although a matrix is 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 $|A^{-1}b - \tilde{x}|$. It is slightly better than expected by the well accepted rule of thumb that the error should be of size $\mathbf{u}\cdot\mathrm{cond}(A)$. That may be due to the sparseness of the input matrices.

TABLE 3
Detailed results for verified inclusion $A^{-1}b \in \tilde{x} \pm \delta$ by residual iteration

		symmetric		general	
1	# in [10]	1210		438	
2	n	20360		1633	
3	nnz(A)	509866		46626	
4	$\sigma_{\min}(A)$	7.9e-15		2.1e-10	
5	$\|\tilde{x}\|_\infty$	1.000		0.999	
6	$\|A\tilde{x} - b\|_\infty$	8.4e-16		3.0e-16	
7	error $\tilde{x}$	2.3e-6	0.029	2.6e-9	6.9e-6
8	$\|\tilde{y}\|_\infty / \|\tilde{x}\|_\infty$	2.3e-6		7.3e-8	
9	$\|A\tilde{x} + A\tilde{y} - b\|_\infty$	5.5e-21		3.7e-24	
10	error $\tilde{x} + \tilde{y}$	2.2e-9	2.6e-5	3.9e-17	2.0e-13
11	$\|\tilde{z}\|_\infty / \|\tilde{x}\|_\infty$	2.2e-9		5.6e-16	
12	$\|A\tilde{x} + A\tilde{y} - b\|_\infty$	5.6e-24		2.6e-32	
13	error $\tilde{x} + \tilde{y}$	2.0e-12	2.4e-8	1.9e-17	5.2e-17
14	$\varrho = \|A\tilde{x} + A\tilde{y} - b\|$	4.2e-28	5.6e-24	9.3e-34	2.6e-32
15	$\delta = \|\tilde{y}\| + \|\varrho\|_2 / s$	5.4e-9	5.4e-9	1.5e-17	5.5e-17
16	entrywise accuracy of inclusion	1.1e-8	1.3e-4	3.9e-17	1.0e-16

The next Line 3 in Algorithm ErrorBound in Table 2 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 2 makes sure that the bit representations of $\tilde{x}$ and $\tilde{y}$ do not overlap. The transformation TwoSum is error-free, the sum does not change.

The $\tilde{y}$ in Line 8 is the value computed in Line 3 of ErrorBound, so the unevaluated sum $\tilde{x} + \tilde{y}$ covers about 22 or 24 digits, respectively. As shown in Lines 9 and 10 of Table 3 the unevaluated sum $\tilde{x} + \tilde{y}$ has a smaller residual, corresponding to the overlap of $\tilde{x}$ and $\tilde{y}$, and better accuracy. By the cited rule of thumb the improvement should be of the order $\mathbf{u}\cdot\mathrm{cond}(A)$, but in both examples it is a little 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 $\tilde{y}$. That is indeed true as by Line 11. It follows that the unevaluated sum $\tilde{x} + \tilde{y} + \tilde{z}$ spans about 25 or 32 figures, respectively. The reason that no triple precision is achieved is that

the residuals are calculated in twice, not thrice the working precision. Therefore, it would be no advantage to keep the three parts but $\tilde{z}$ is added to $\tilde{y}$. For the new approximation $\tilde{x}+\tilde{y}$, Line 6 in ErrorBound ensures again that the bits don't overlap.

As by Lines 12 and 13 in Table 3 this approximation has again smaller residual and improved accuracy. Correspondingly, the upper bound ϱ on $|A\tilde{x}+A\tilde{y}-b|$ 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 $\alpha := \|\varrho\|_2/\sigma_{\min(A)}$, i.e., $|A^{-1}b-(\tilde{x}+\tilde{y})| \leqslant \alpha$.

By combining the error bound into the vector $\delta = |\tilde{y}| + \|\varrho\|_2/s$ 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 3 shows the median and maximum accuracy of the inclusion in terms of the relative error $|\delta_k/\tilde{x}_k|$. In the first example, due to the condition number, some 4 to 8 decimal figures of the left and right bounds coincide. In the second example the bounds are maximally accurate. Repeating the residual iteration in steps 5 and 6 of Algorithm ErrorBound in Table 2 another 3 times yields almost maximally accurate results for all entries of the first example as well.

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 [39, 52]. In order to compute error bounds for $\Sigma(\mathbf{A},\mathbf{b})$ we use a midpoint-radius representation for $\mathbf{A}$. The INTLAB commands `M = mid(A)` and `R = rad(A)` compute matrices $M, R \in \mathbb{F}^{n\times n}$ with $M-R \leqslant A \leqslant M+R$ for all $A \in \mathbf{A}$, and similarly for $\mathbf{b}$.

For interval input, there is no gain when using an extra precise residual iteration as in Algorithm ErrorBound in Table 2. Denote by $\tilde{x}$ an approximate solution of the midpoint linear system $Mx = \mathrm{mid}(b)$ after few residual iterations. Let $A \in \mathbf{A}, b \in \mathbf{b}$ fixed but arbitrary. For the moment assume that $\mathbf{b}$ is an interval vector, i.e., $\mathbf{b} \in \mathbb{IF}^n$. Denote $\Delta := A - M$. Then $|\Delta| \leqslant R$ and we adapt (4.1) into

$$\begin{aligned}
|A^{-1}b-\tilde{x}| &= |(M+\Delta)^{-1}(b-A\tilde{x})| \\
&= |(I+M^{-1}\Delta)^{-1}M^{-1}(b-A\tilde{x})| \\
&\leqslant \|(I+M^{-1}\Delta)^{-1}M^{-1}(b-A\tilde{x})\|_\infty \mathbf{e} \\
&\leqslant \|(I+M^{-1}\Delta)^{-1}M^{-1}(b-A\tilde{x})\|_2 \mathbf{e} \\
&\leqslant \frac{\|M^{-1}(b-A\tilde{x})\|_2}{1-\|M^{-1}\Delta\|_2}\,\mathbf{e} \\
&\leqslant \frac{\sigma_{\min}(M)^{-1}\|\mathbf{b}-\mathbf{A}\tilde{x}\|_2}{1-\sigma_{\min}(M)^{-1}\|R\|_2}\,\mathbf{e} \\
&\leqslant \frac{\|\mathbf{b}-\mathbf{A}\tilde{x}\|_2/s}{1-\|R\|_2/s}\,\mathbf{e}
\end{aligned} \tag{5.2}$$

which is true provided that $0 < s \leqslant \sigma_{\min}(M)$ and $\|R\|_2/s < 1$. For multiple right hand sides, i.e., $\mathbf{b} \in \mathbb{F}^{n,k}$ with $k > 1$, apply (5.2) successively to the columns of $\mathbf{b}$.

The estimates (4.1) for point data and (5.2) for interval data are the key to the inclusion of the solution of $Ax = b$. The necessary switch from the ∞-norm of $b - A\tilde{x}$ to the 2-norm introduces an unfortunate weakening of the bound. For point systems that can be compensated by storing the approximation as an unevaluated sum and accurate evaluation of the residual, so that in principle the bound for the residual can be arbitrarily small. Unfortunately that is not possible for interval systems, and this will become relevant for nonlinear systems to be discussed in Part II of this note.

Note that the successful computation of a positive lower bound s of $\sigma_{\min}(M)$ together with $\|R\|_2/s < 1$ verifies the nonsingularity of every $A \in \mathbf{A}$ 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 M and/or smaller radius R.

6. Symmetric (positive definite) matrices. As has been mentioned, "positive definite" is in parenthesis because this is no assumption on the input matrix but will be proved a posteriori by our algorithm. The corresponding algorithm "verifySparseSPD1" will be improved in Part II to accept an unsymmetric input matrix. Moreover, it will also be shown in Part II that Algorithm "verifySparseSym1", to be introduced in Section 8 of this note, works correctly for unsymmetric input matrix as well. Therefore we use "verifySparseSPD1", "verifySparseNearSym1" and "verifySparseGen1" for the versions in this Part I, and accordingly "verifySparseNearSPD2", "verifySparseNearSym2" and "verifySparseGen2" in Part II.

THEOREM 6.1. *Let symmetric $A \in \mathbb{F}^{n\times n}$ and $0 < s \in \mathbb{F}$ be given. Denote $B := float(A - sI)$ computed in rounding downwards. Suppose the floating-point Cholesky decomposition of B runs to completion producing a Cholesky factor $\tilde{R}$. Define $\Delta B := \tilde{R}^T\tilde{R} - B$. Then*

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

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 $\Phi \in \mathbb{R}^{n\times n}$ the matrix with $\Phi_{ij} := \min(\mu_i, \mu_j) + 1$ and by $D \in \mathbb{R}^{n\times n}$ the diagonal matrix with $D_{kk} = \left(\frac{B_{kk}}{1-\Phi_{kk}\mathbf{u}}\right)^{1/2}$. Let

$$\mathbf{u}\|D\Phi D\|_2 \leqslant \alpha \tag{6.2}$$

for α computed by Corollary 2.12 and assume $s \geqslant \alpha$. Then $\|\Delta B\|_2 \leqslant \alpha$ and

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

if the decomposition was performed using nearest operations. If $4(\max\mu_k + 1)\mathbf{u} \leqslant 1$, then (6.3) remains true for faithful operations when using $\Phi_{ij} := \min(\mu_i, \mu_j) + 2$ and 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 2.10. Denote $\Delta := A - sI - B$. Then Δ is nonnegative, and (1.11) yields

$$\begin{aligned} \lambda_{\min}(A) - s &= \lambda_{\min}(A - sI) = \lambda_{\min}(B + \Delta) \\ &\geqslant \lambda_{\min}(B) = \lambda_{\min}(\tilde{R}^T\tilde{R} - \Delta B) \geqslant -\|\Delta B\|_2 \geqslant -\alpha \end{aligned}$$

and proves $\lambda_{\min}(A) \geqslant s - \alpha \geqslant 0$, and therefore $\lambda_{\min}(A) = \sigma_{\min}(A)$ and (6.3). The assertion for faithful operations follows as in Lemma 2.10. ∎

1 function $[x,\delta]$ = verifySparseSPD1(A,b)
2 If any $A_{kk}A_{\ell\ell} \leqslant 0, [x,\delta]$ = verifySparseNearSym1(A,b), return
3 If $A_{11} < 0, A := -A;\ b := -b;$ % A possibly negative definite
4 Equilibrate A by (3.1)
5 Compute Cholesky factorization $\tilde{R}^T\tilde{R} \approx A$ by (3.4)
6 If failed, $[x,\delta]$ = verifySparseNearSym1(A,b), return
7 Compute $\tilde{s}(\tilde{R})$ by (3.9) and set $s := 0.9\tilde{s}$
8 If $s < 10^{-16}\|A\|_\infty$, verif. fails, $[x,\delta]$ = verifySparseNearSym1(A,b), return
9 `setround(-1); As = A - s * speye(n);`
10 `setround(0); [Rs,FLAG,p] = chol(As);`
11 If succeeded, go to Step 15
12 Set $s := s/5$ and compute in rounding downwards $As = A - sI$
13 Compute Cholesky factor $Rs^TRs \approx As$ in rounding to nearest by (3.4)
14 If failed, $[x,\delta]$ = verifySparseNearSym1(A,b), return
15 Compute upper bound $\alpha :=$ r.h.s.(6.2) with $\|Rs^TRs - As\|_2 \leqslant \alpha$
16 If $\alpha \geqslant s$, compute α with $\|Rs^TRs - As\|_2 \leqslant \alpha$ using (2.11)
17 If $\alpha \geqslant s$, compute α with $\|Rs^TRs - As\|_2 \leqslant \alpha$ using (2.13)
18 If $\alpha \geqslant s$, verification failed, return
19 $[x,\delta]$ = ErrorBound$(A, b, s - \alpha,$ "*solve*") using $\tilde{R}$ for solve

TABLE 4
Verified error bounds for $A^{-1}b$ for symmetric (positive definite) sparse input matrix A

In Table 4 we sketch our subalgorithm "verifySparseSPD1" for solving a sparse linear system with symmetric (positive definite) matrix. As has been mentioned, the mathematical assumption is that the input matrix A is symmetric. If A is indefinite and/or positive definiteness cannot be verified, then our subalgorithm ""verifySparseNearSym1"" for symmetric input matrix as given in the next section is called.

The details of subalgorithm "verifySparseSPD1" are as follows. If there are diagonal elements with nonpositive product, A can neither be positive nor negative definite and we call subalgorithm "verifySparseNearSym1". Reaching Step 3 all diagonal entries must have the same sign, and (A, b) is replaced by $(-A, -b)$ if they are negative. Otherwise, after equilibration in Line 4 a numerical Cholesky decomposition `[R,FLAG,p] = chol(A,'vector')` is computed in Line 5. If `FLAG` $\neq 0$, the factorization failed and subalgorithm "verifySparseNearSym1" is called. Otherwise, an approximative lower bound s of the smallest singular value of A is computed by (3.9) in Line 7.

If s in Line 7 is too small to hope for a successful inclusion, we call directly "verifySparseNearSym1" in Step 8. Next in Line 9 a lower bound `As` of the shifted matrix $A - sI$ is computed and Theorem 6.1 is applicable. Next, a floating-point Cholesky decomposition of `As` is tried in Line 10. In case of failure we try again with a smaller value for s.

The potential second decomposition in Line 13 may fail because of ill-conditioned input matrix A or, if s is still too large. In that case we call subalgorithm "verifyS-

parseNearSym1". In the next Line 15 an upper bound α as in (6.2) in Theorem 6.1 is computed using the code in Corollary 2.12 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 and works very often. 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 in Step 16 by initializing `setround(1), E = Rs'*Rs-As;` and using (2.11) in Lemma 2.5. That bound is almost always sufficiently good. If still $\alpha \geqslant s$, we invest some computing time and improve α again by computing ΔB in extended precision with rounding to nearest and using (2.13).

This is our last resource. It still $\alpha \geqslant s$, "verifySparseSPD1" failed to compute verified error bounds, and our general Algorithm `verifySparselss1` to be presented in Table 7 calls subalgorithm "verifySparseNearSym1". 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 2. This algorithm requires to solve a linear system $Ay = c$ for some right hand side c which is performed using $\tilde{R}$ in Line 5.

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

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

For the anticipated decomposition of B we distinguish three cases. If B is 1×1, then $B = b$ and $M := \sqrt{|b|}$, $S = \mathrm{sign}(b)$ and $P = 1$ do the job.

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

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

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

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

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 might occur.

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 $\widehat{D}$, real signature matrix S and permutation matrix P. Once Matlab's `ldl` works for complex sparse matrices as well, the above approach is easily adapted with real signature matrix S to ensure that the multiplication by SP is error-free.

8. Symmetric matrices. We show in Table 5 a general outline of our subalgorithm "verifySparseNearSym1", intended to compute verified bounds for the solution of a sparse linear system with symmetric matrix. Surprisingly, as will be shown in Part II of this note, this algorithm works correctly for unsymmetric input matrix A as well. That explains the unusual naming. In principle, the algorithm works for general input matrix, however, success is only likely for nearly symmetric matrices, i.e., $\|A^H - A\|$ small compared to $\|A\|$.

After equilibration in Line 2 we decompose A in Line 3. Next L_1, L_2 are computed in Steps 5 – 6 with $A \approx L_1L_2$. The factors are computed in floating-point, but because S is a real 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 (1.12) 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 $M = \mathrm{float}(L_1L_1^T)$ 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 17 to compute a true lower bound on $\sigma_{\min}(M)$ leading to a lower bound for $\sigma_{\min}(A)$. If s is so small that there is no hope for a verification, we immediately call "verifySparseGen1" in Step 9.

Aiming for a correct lower bound on $\sigma_{\min}(A)$ we compute an upper bound α on $\|A - L_1L_2\|_2$ in Line 10. If α is not small enough, i.e., $\alpha \geqslant s$, then α is improved by (2.11) in Line 11. Next we use (2.9) to compute an upper bound β on $\|M - L_1L_1^T\|_2$. 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 invest some time in Line 14 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 "verifySparseGen1" is called in Step 15.

In Line 16 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 17. If not successful, $\widehat{M}$ and s are updated as in Lines 12 – 13 of

1 function $[x,\delta]$ = verifySparseNearSym1(A,b)
2 Equilibrate A by (3.2)
3 Compute $LDL^T(A)$ by (3.5)
4 If D is singular, $[x,\delta]$ = verifySparseGen1(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 $M \approx L_1 L_1^T$ in rounding upwards
8 Compute Compute $\tilde{s}(L_1)$ by (3.9) and set $s := 0.9\tilde{s}$
9 If $s < 10^{-16}\|A\|_\infty$, verification fails, $[x,\delta]$ = verifySparseGen1(A,b), return
10 Use (2.10) to compute α with $\|A - L_1L_2\|_2 \leqslant \alpha$
11 If $\alpha \geqslant s$, improve α as in (2.11)
12 If $\alpha < s$, use (2.9) to compute β with $\|M - L_1L_1^T\|_2 \leqslant \beta$, else $\beta = \infty$
13 If $\alpha < s$ and $\alpha + \beta \geqslant s$, improve β as in (2.11)
14 If $\alpha + \beta \geqslant s$, recompute M and improve α, β as in (2.13)
15 If $\alpha + \beta \geqslant s$, $[x,\delta]$ = verifySparseGen1(A,b), return
16 Compute $\widehat{M} := M - sI$ in rounding downwards
17 Compute Cholesky factor $\tilde{R}^T\tilde{R} \approx \widehat{M}$ in rounding to nearest by (3.4)
18 If succeeded, go to Step 22
19 Set $s := s/5$ and compute in rounding downwards $\widehat{M} = M - sI$
20 Compute Cholesky factor $\tilde{R}^T\tilde{R} \approx \widehat{M}$ in rounding to nearest by (3.4)
21 If failed, $[x,\delta]$ = verifySparseGen1(A,b), return
22 Compute γ with $\|\widehat{M} - \tilde{R}^T\tilde{R}\|_2 \leqslant \gamma$ by (6.2) in rounding upwards
23 If $\alpha + \beta + \gamma \geqslant s$, improve γ as in (2.11)
24 If $\alpha + \beta + \gamma \geqslant s$, improve γ as in (2.13)
25 If $\alpha + \beta + \gamma \geqslant s$, $[x,\delta]$ = verifySparseGen1(A,b), return
26 $[x,\delta]$ = ErrorBound$(A, b, s - \alpha - \beta - \gamma,$ "*solve*") using LDL^T for solve

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

"verifySparseSPD1", and for the smaller shift s a Cholesky decomposition is tried in Line 20.

If the second decomposition is still not successful, then the verification failed and subalgorithm "verifySparseGen1" is called in Step 21. Otherwise, an upper bound γ using the a priori bound in (6.2) is computed in Line 22 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 $\sigma_{\min}(M) \geqslant s - \gamma$. In fact, here some care is necessary. In order to apply Theorem 6.1 the matrix $\widehat{M} = \mathrm{float}(L_1L_1^T - sI)$ has to be symmetric. However, $\mathrm{float}(L_1L_1^T)$ is computed in floating-point, and blocking strategies or other measures to improve the performance might produce an unsymmetric matrix.[13] Fortunately, that cannot happen in Matlab.

[13]It is easy to construct examples where changing an off-diagonal entry by 1 bit causes a symmetric positive matrix to become indefinite.

Here the statement `M = L1*L1'` is analysed, and in order to improve the performance only the lower half is computed and mirrored. This is true for real and complex data and in any rounding mode. In another environment we have to replace M by its symmetrized version $(M^T + M)/2$.

If the sum $\alpha + \beta + \gamma$ of errors is too large, then the verification failed and we turn to subalgorithm "verifySparseGen1" in Step 25. Otherwise, i.e., $\alpha + \beta + \gamma < s$, (1.10), (8.1) and Theorem 6.1 yield

$$\begin{aligned} \sigma_{\min}(A) &\geqslant \sigma_{\min}(L_1 L_2) - \|A - L_1 L_2\|_2 \geqslant \sigma_{\min}(L_1 L_1^T) - \|A - L_1 L_2\|_2 \\ &\geqslant \sigma_{\min}(M) - \|L_1 L_1^T - M\|_2 - \|A - L_1 L_2\|_2 \geqslant \sigma_{\min}(M) - \beta - \alpha \\ &\geqslant s - \alpha - \beta - \gamma \ . \end{aligned} \tag{8.2}$$

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 2. Note that the conclusion remains valid when replacing M by $(M^T + M)/2$. In fact, the first two lines in (8.2) are valid for any matrix M, and the last line for any symmetric M. In Part II of this note we will show that the algorithm works correctly for unsymmetric matrices as well.

9. General matrices. As in [50, 61] 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 $\pm$ 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 each 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 shows 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 [17, 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 $(1,\ldots,2n)$ to $(k,m,1,\ldots,k-1,k+1\ldots,m-1,m+1,\ldots,2n)$. 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 [48, 50, 61] we proceed in "verifySparseGen1" in Table 6 for general matrices as follows. After equilibrating the original matrix A we compute an LDL^T-decomposition of the augmented matrix B in Step 4 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 should have exactly $2n$ nonzero entries for nonsingular A. If the decomposition fails, i.e., there are less than $2n$ nonzero elements in D, we use the LDL^T-decomposition as in (3.7). That happened a number of times, but fortunately the bug is removed from Matlab Version 2026a on.

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 real signature and a permutation matrix. That computation is error-free, so that, as in subalgorithm "verifySparseNearSym1", 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). The remaining of the subalgorithm until Line 26 is, *mutatis mutandis*, identical to subalgorithm `verifySparseNearSym1` in Table 5, 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

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

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".

```
function [x, δ] = verifySparseGen1(A,b)
    Equilibrate A by (3.3)
    Let B the augmented matrix (9.1)
    Compute LDL^T(B) by (3.5)
    If nnz(D) < 2n, compute LDL^T(B) by (3.7)
    If nnz(D) < 2n, verification failed, return
    Compute approximate splitting D ≈ D̂SPD̂^T according to (7.2)
    Compute L1 ≈ LD̂ and L2 = SPL1^T
    Compute M ≈ L1 L1^T in rounding upwards
    Compute s̃(L1) by (3.9) and set s := 0.9s̃
    Use (2.10) to compute α with ‖B − L1 L2‖2 ≤ α
    If α ≥ s, improve α as in (2.11)
    If α < s, use (2.9) to compute β with ‖M − L1 L1^T‖2 ≤ β, else β = ∞
    If α < s and α + β ≥ s, improve β as in (2.11)
    If α + β ≥ s, recompute M and improve α, β as in (2.13)
    If α + β ≥ s, verification failed, return
    Compute M̂ := M − sI in rounding downwards
    Compute Cholesky factor R̃^T R̃ ≈ M̂ in rounding to nearest by (3.4)
    If succeeded, go to Step 23
    Set s := s/5 and compute in rounding downwards M̂ = M − sI
    Compute Cholesky factor R̃^T R̃ ≈ M̂ in rounding to nearest by (3.4)
    If Cholesky decomposition ends premature, verification failed, return
    Compute γ with ‖M̂ − R̃^T R̃‖2 ≤ γ by (6.2) in rounding upwards
    If α + β + γ ≥ s, improve γ as in (2.11)
    If α + β + γ ≥ s, improve γ as in (2.13)
    If α + β + γ ≥ s, verification failed, return
    [x, δ] = ErrorBound(B, [0; b], s − α − β − γ, "solve") using LDL^T for solve
```

TABLE 6
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] = verifySparselss1(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] = verifySparseSPD1(A,b);
      end
      if ( ~symm ) || isnan(xs(1))   % A unsymm. or SPD failed
        [xs,delta] = verifySparseGen1(A,b);
      end
    else                    % A real, b complex
      [xs,delta] = verifySparselss1(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] = verifySparselss1(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 verifySparselss1
```

TABLE 7
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 7 the self-contained Algorithm `verifySparselss1` as executable Matlab/INTLAB code to solve a general real or complex sparse linear systems. The second and final version of Algorithm `verifySparselss1` including least squares problems and underdetermined linear systems will be given in Table 11 in Part II of this note including handling unsymmetric matrices by "verifySparseNearSPD2" and "verifySparseNearSym2".

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 check that all diagonal elements of A have the same sign or some Cholesky decomposition fails, then "verifySparseSPD1" calls subalgorithm "verifySparseNearSym1". If it fails as well, then as a final resource subalgorithm "verifySparseGen1" is called.

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 [61]. For a linear system $Ax = b$ the algorithm proposed by Terao and Ozaki [61] is based on Theorem 1.1 to compute a lower bound for $\sigma_{\min}(A)$, basically as in Table 8, with the residual iteration in [56].

If successful, i.e., $\theta > \varrho$, then $\sigma_{\min}(B) = \sigma_{\min}(A) > \theta - \varrho$. The Matlab code is published in [61] and some missing code was kindly provided by the authors. In [61] 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 $x = A^{-1}b$, where we use (9.2). The advantage of their method

TABLE 8
Computation of a lower bound $\theta - \sigma$ for $\sigma_{\min}(A)$

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 $(n, n, 0)$, 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

compared to Theorem 1.1 in [48] is that only one decomposition, namely of $B + \theta I$ is necessary because for nonsingular A the inertia $(n, n, 0)$ of B is known beforehand. A drawback is that for spd and symmetric matrices the augmented matrix B of double size is decomposed, and that only a decomposition of the shifted matrix $B + \theta I$ is available, not of B. They use the residual iteration in [56] based on the factorization of $B + \theta I$. As analysed in [56], this iteration converges if $\theta < \frac{1}{2}\sigma_{\min}(B)$, and the residual decreases by the factor $\frac{\theta}{1-\theta}$ in each step. This technique used in [61] is the reason why θ is set to $\frac{1}{2}s$ in Step 1 in Table 8.

Another drawback is that this limits the applicability of the method in [61]. In Figure 1 we use matrix number 358 in [10], replace it by $M := A - sI$ and calculate the shift s such that M has the condition number as displayed. To avoid numerical instabilities, we calculate M in double precision but the 2-norm condition number of M using extended precision in Advanpix [18]. The maximal condition number for which [61] computes an inclusion is $\kappa_2 = 1.4 \cdot 10^{12}$, whereas `verifySparselss1` is successful up to $\kappa_2 = 2.0 \cdot 10^{15}$.

Moreover, when θ is less than but close to $\frac{1}{2}\sigma_{\min}(B)$ the convergence of the residual iteration slows down. In our examples to be presented in the next section

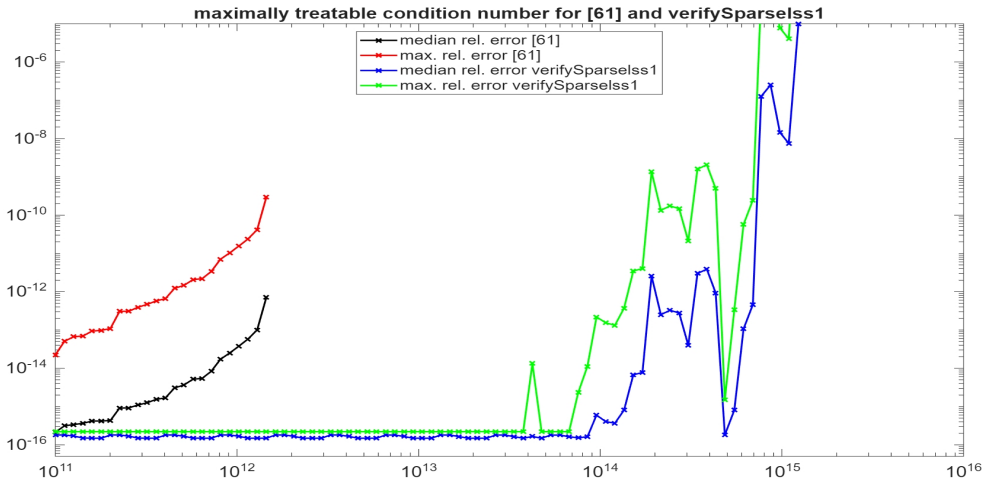

FIG. 1. *Maximally treatable condition number for [61] and* `verifySparselss1`

there was a case (number 46 in [10]) were the residual iteration in [61] diverged because θ was slightly too large. That may happen due to rounding errors. In that case the verification was successful but the inclusion of all entries $[-10^{206}, +10^{206}]$. Thus a better choice might be $\theta := 0.4s$ which, however, reduces the maximal treatable condition number further.

A difficulty of Theorem 1.1 and therefore of [61] is that an inclusion of the product of three matrices is needed. In [61] the code

$$\texttt{[L,D,p] = ldl(mid(G),'vector'); rho = NormBnd(G(p,p) - L * intval(D) * L',true);}$$

computes ϱ with $\|B + \theta I - LDL^T\|_2 \leqslant \varrho$ and uses `NormBnd` from Table 1. 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 further the maximal possible condition number until which a verification is possible.

Another problem slowing down [61] is that the decomposition of the shifted matrix B causes significantly more fill-in than the decomposition of the original augmented matrix B. We see this effect in Figure 5 of Part II of this note.

The algorithm in [61] is called by

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

with additional parameters `precond` and `acc`. The output `X` is an interval vector, and if successful, $A^{-1}b \in$ `X`. The meaning of `acc` is as follows. When multiplying two interval matrices, there is a choice in INTLAB [49] between using midpoint-radius and infimum-supremum arithmetic. 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 in midpoint-radius representation uses Level-3-BLAS routines and is much faster than infimum-supremum using rank-1 updates [53]. 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 methods for interval matrix multiplication differ only if both factors comprise of intervals with nonzero diameter. The most important product in

the code of [61] in Table 8 is `L*intval(D)*L'`, but here and elsewhere at least one factor is a point matrix. Therefore there is no difference between the two methods in INTLAB for multiplication. Consequently, we observed, if any, a marginal difference between the quality of the bounds and timing using $false$ or $true$ for `acc`, which is confirmed by the test results in [61]. 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 8 an equilibration similar to (3.3) is applied. Switching `precond` on or off has significant influence on the performance and accuracy of the algorithm in [61]. 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 but success without preconditioning. In our computational results we found 7 such cases and appended the corresponding computing time by an "*".

Another reason to use `precond` = $true$ for the algorithm in [61] is that when using `precond` = $false$ the inclusion may be wide due to problems with the residual iteration as explained above. For instance, in example 1404 the verified inclusion by [61] with `precond` = $false$ ends successfully, but all entries of the inclusion are equal to $[-4.45\cdot 10^{17}, 4.45\cdot 10^{17}]$. With equilibration [61] could guarantee at least 11 figures.

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 the Matlab prerelease 2026a [35] under Windows 10.

As has been mentioned, [61] uses the Advanpix package [18] for accurate residual calculations. In Part II we introduce an alternative algorithm `spProdK` written in pure Matlab code which is included in INTLAB [49]. Advanpix respects the rounding mode only in extended precision, while `spProdK` allows to specify higher precisions. All results of our Algorithm `verifySparselss1` (henceforth called "Sparse1") presented in Part I of this note use Advanpix to allow a fair comparison to [61], where in Part II the algorithm is the same but uses `spProdK`. Therefore, results of `verifySparselss1` for the same test example may differ between Part I and Part II. Because of the possibility to specify higher than extended precision, the results of Sparse1 using `spProdK` in Part II are generally more accurate than using Sparse1 with Advanpix in this Part I.

As for test matrices we used the Suite Sparse Matrix Collection [10] with the interface [24]. More precisely, we treat all 395 square matrices with dimension

$$10^3 \leqslant n \leqslant 10^5 \quad \text{and} \quad 10^7 \leqslant \kappa_2(A) \leqslant 10^{16} \quad \text{and} \quad \texttt{nnz(A)} \leqslant 10^6 \ . \tag{12.1}$$

One goal of the tests is to check whether our method and that in [61] work up to the limit condition number $\kappa_2(A) \lesssim \mathbf{u}^{-1} \approx 10^{16}$. Since both methods rely on a lower bound of the smallest singular value of A, the 2-norm condition number $\kappa_2(A) = \|A^{-1}\|_2\|A\|_2$ is appropriate rather than `condest(A)` which estimates the 1-norm condition number. The latter is, in general, larger.

In Part I of this note we consider only real test cases satisfying (12.1) because the algorithm in [61] does not handle complex matrices. After removing 3 complex and 4 diagonal matrices 388 test cases remain. In [61] 20 test cases are considered, 8 of which satisfy (12.1). We keep the tests from [61] together so that we arrive at totally 400 test cases where 35 are spd, 96 are symmetric, 249 are general plus the 20 tests taken from [61].

In addition we consider 100 random test cases with matrices of dimension $n = 10^4$,

TABLE 9
Timing T and inclusion X for algorithms Sparse1 and [61]

```
tic     % compute inclusion by Sparse1 or [61] with equilibration
equil = true; X = Inclusion(A,b,equil);
T = toc;
if isnan(X)   % verification with equilibration failed
  tic         % compute inclusion by Sparse1 or [61] without equilibration
  equil = false; X = Inclusion(A,b,equil);
  if ~isnan(X)
    T = T + toc;
  end
end
```

density 0.001 and condition number about 10^{15}, i.e., generated by

$$\texttt{A} = \texttt{sprand}(\texttt{n},\texttt{n},\texttt{dens},1/\texttt{cnd}) \quad \text{with } \texttt{n} = 10^4,\ \texttt{dens} = 0.001 \text{ and } \texttt{cnd} = \texttt{1e15}. \tag{12.2}$$

In the median the random matrices have 95,020 nonzero elements and 2-norm condition number $1.90 \cdot 10^{15}$. We define the relative error of an interval $\mathbf{X} \in \mathbb{IR}$ by

$$\text{relerr}(\mathbf{X}) := \begin{cases} \text{rad}(\mathbf{X})/\text{mid}(\mathbf{X}) & \text{if } 0 \notin \mathbf{X} \\ \text{rad}(\mathbf{X}) & \text{otherwise .} \end{cases} \tag{12.3}$$

We use that definition to avoid the usual difficulties with intervals being very narrow but enclosing zero. An inclusion is called "poor" if the relative error exceeds 0.01, i.e., barely one digit is guaranteed. For the verification algorithms we consider the median and maximum of the relative errors of the inclusions of all entries. The inclusions of our Algorithm `verifySparselss1` are almost always maximally accurate. Thus we say that for $\tilde{x}$ computed by Matlab's "backslash" an entry $\tilde{x}_i$ is a poor approximation if the relative error to the corresponding entry of the midpoint result of Sparse1 exceeds 0.01.

We compare our Algorithm Sparse1 with the algorithm in [61]. For both algorithms using the equilibrated matrix is the method of choice because in most cases the condition number decreases, often substantially. However, as will be seen in Section 10 in Part II of this note, in rare cases the condition number increases for the equilibrated matrix. Because we are interested in verified inclusions of the solution, we use for both algorithms the method as in Table 9: The timing and result of the algorithm with equilibration is used if successful. If not successful, the algorithm without equilibration is tried. If without equilibration not successful as well, for both [61] and Sparse1 only the timing of the first call is used because we consider the call with equilibration as the default. If successful without equilibration, that result and the sum of the computing times of both calls is used and displayed later with an "$*$" appended. For the 400 test cases from [10] that happened 7 times for the algorithm in [61] but never for Sparse1. For the random tests it never occurred for both algorithms.

An overview of all results is displayed in Table 10. The first column indicates the structure, namely symmetric positive definite, symmetric indefinite, general, all test matrices out of [61] and the random tests according to (12.2). In two cases we observed failure of Matlab's "backslash". In example 1417 from [10] 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.

TABLE 10
Results for all real samples satisfying (12.1) *or* (12.2)

	performance "backslash"				performance Sparse1				performance [61]			
structure	poor	fail			poor	fail			poor	fail		
spd	2	0	of	35	0	0	of	35	1	8	of	35
sym	2	0	of	96	0	0	of	96	0	7	of	96
gen	1	0	of	249	2	1	of	249	15	12	of	249
tests [61]	0	2	of	20	0	0	of	20	0	0	of	20
random	35	0	of	100	0	4	of	100	27	38	of	100

Our Algorithm Sparse1 fails in 1 out of the 400 [10] test cases, the Algorithm in [61] failed 27 times. For the random samples as in the last line of Table 10 the Algorithm in [61] failed in 38 out of 100 tests, and 27 out of the 62 successful inclusions of the solution vector contained poor inclusions for some entries. Our Algorithm Sparse1 computed verified bounds in 96 out of the 100 random test cases with no poor inclusions. As can be seen there are also poor approximations by Matlab's "backslash", in particular for the random tests.

Before coming to the numerical results we show in Figure 2 the benefit of the new a priori estimate (2.15) together with Corollary 2.12 for the residual $\|\tilde{R}^T\tilde{R} - A\|_2$ of the Cholesky decomposition compared to (2.19) (cf. [56, Lemma 2.2]). These a priori estimates are used in Step 15 in "verifySparseSPD1", Step 22 in "verifySparseNearSym1" and Step 23 in "verifySparseGen1". The a priori bounds come at low cost, and if not sufficiently good, more time consuming measures are taken.

Test cases are the matrices As in Step 15 of "verifySparseSPD1" for the 35 symmetric positive definite matrices in [10] satisfying (12.1), sorted by increasing dimension. Between 5 to 9 power iterations were used to improve (2.15), in the median 6 iterations. The ratio of the bounds (2.19) divided by (2.15) ranges between 21 and 1516, in the median the new bound is better than (2.19) by the factor 121.

Next we show detailed results. The dimension, number of nonzero elements and condition number of all test cases is shown in Figure 3. The dimensions vary between $1,019$ and $682,862$ and the number of nonzero elements between $3,699$ 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 [61] the right hand side `A*ones(n,1)` was used. In [61] computational results are listed for the four options `acc` and `precond` *true* and $false$, but no recommendation was given which combination to use. As explained above, the parameter `acc` has no influence at all, and concerning the parameter `precond`, we proceed as in Table 9.

In Figure 4 we show for all tests the ratio of computing times of Matlab's "backslash" divided by that for our new Algorithm Sparse1, and for the algorithm in [61] divided by Sparse1. Ratios are not displayed if one algorithm fails. A number less than 1 in the left graph means that "backslash" is faster than Sparse1, and a number larger than 1 in the right graph means that Sparse1 is faster than [61]. In principle we expect that obtaining an approximation by Matlab's "backslash" should be faster than computing a verified inclusion by Sparse1. However, Figure 4 shows that Sparse1 is often faster than "backslash", see Table 11 for details.

In the median over all 400 examples Matlab's "backslash" is faster than Sparse1

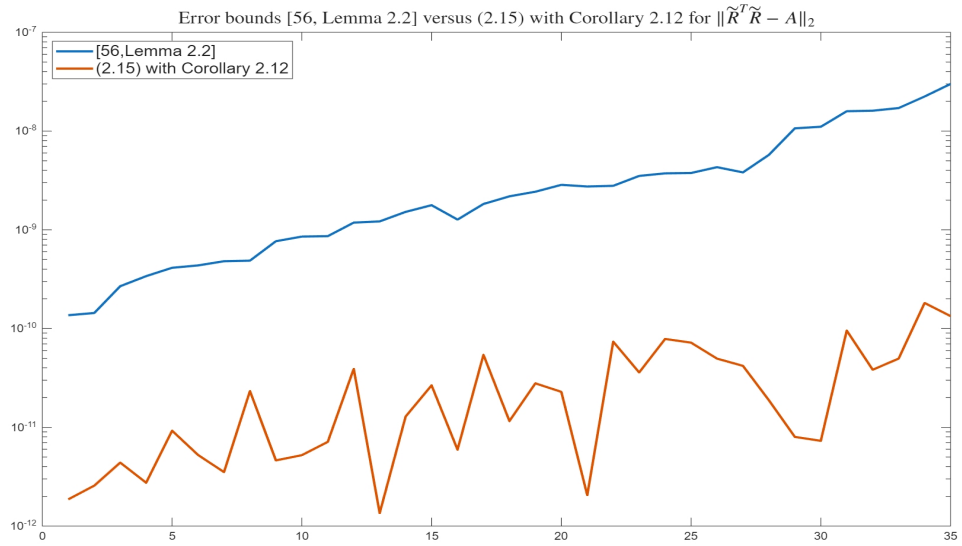

FIG. 2. *Comparison of a priori error bounds for the residual* $\|\tilde{R}^T\tilde{R} - A\|_2$ *of the Cholesky decomposition*

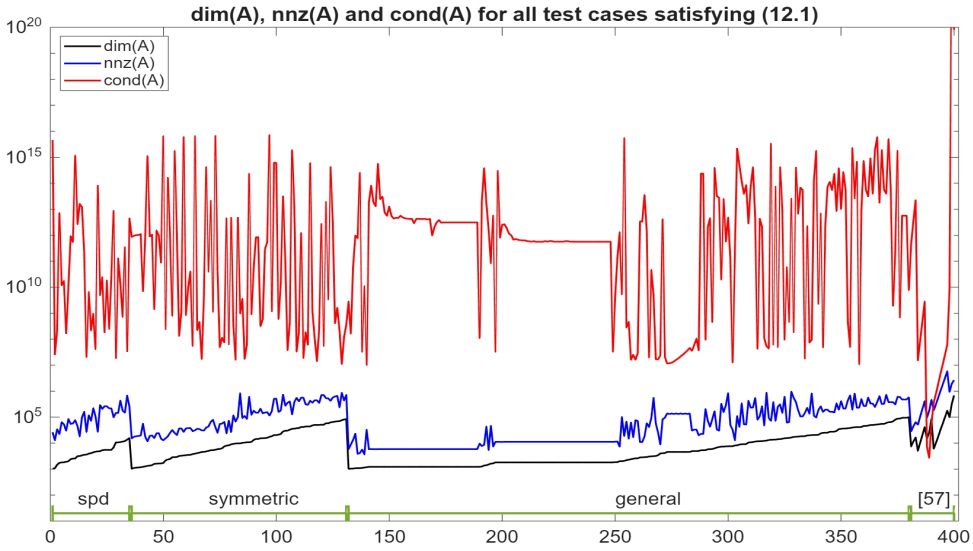

FIG. 3. *Dimension, number of nonzero elements and condition number of all test matrices*

by a factor 3.4. But in 42 out of the 400 cases "backslash" is slower than Sparse1 by a factor 10, and in 13 cases even by a factor 100 and more. An extreme example is number 2214 in [10] with Sparse1 being 391 times faster than "backslash". The matrix has dimension $n = 14,454$ with $147,972$ nonzero elements. The number of nonzero elements of the factor L in our algorithm Sparse1 is $430,688$, whereas "backslash" produces factors L, U with more than 100 times more elements, namely $16,300,793$ for L and $47,932,779$ elements in U. That may explain the large computing time.

We hasten to say that Matlab's "backslash" operator is carefully designed based

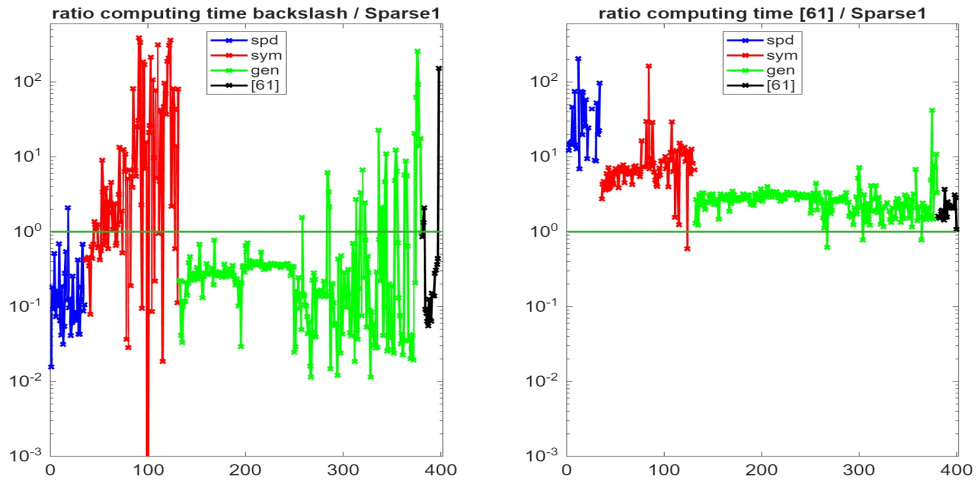

FIG. 4. *Ratios of computing times* $t_{backslash}/t_{Sparse1}$ *and* $t_{[61]}/t_{Sparse1}$

TABLE 11
Detailed time ratios

	ratio $t_{backslash}/t_{Sparse1}$			ratio $t_{[61]}/t_{Sparse1}$		
structure	min	median	max	min	median	max
spd	0.015	0.101	2.090	6.867	22.260	610.043
sym	4.5e-4	3.279	390.683	0.590	6.864	163.198
gen	0.011	0.263	259.165	0.615	2.716	41.823
[61]	0.056	0.145	152.040	1.079	1.997	3.675

on several branches and parameters choosing the appropriate internal algorithm. In some cases it might have been better to choose a different path, however, it is a general purpose method and, as for our Sparse1, the main driving force is robustness. As has been mentioned earlier, there is hardly a general purpose solver for sparse linear systems.

Figure 4 shows that Sparse1 is mostly faster than the algorithm in [61], in the median over all examples by a factor 3.0. In 51 out of the 400 test cases Sparse1 is faster than [61] by a factor 10, and in 4 cases by more than a factor 100. Conversely, there are 4 cases where the algorithm in [61] is faster than Sparse1 with the maximal factor 1.70. In Table 12 the 5 extreme ratios are listed. In the worst case [61] is 610 times slower than Sparse1. Note that [61] failed with equilibration in that example 1606 of [10], but succeeded without.

One reason that the algorithm in [61] is slower than Sparse1 is that the factor L has significantly more fill-in. The reason is that, in contrast to Sparse1, [61] uses the augmented matrix of double size for spd and symmetric matrices, and the shifted augmented matrix for general input matrix. In Figure 5 the ratio of the number of elements in L for the algorithm in [61] versus Sparse1 is shown (for spd matrices we use the Cholesky factor for Sparse1). The ratio is displayed if both algorithms compute a verified inclusion successfully, that explains the gaps. In the median over all 400 examples the L in [61] comprises of 2.24 more elements compared to Sparse1.

Next we show in Figure 6 a rough image of the median relative error of the ap-

TABLE 12
The 5 best and worst time ratios $t_{[61]}/t_{Sparse1}$ out of the 400 test cases

matrix		*times [sec]*		*relerr Sparse1*		*relerr [61]*		
#	n	$t_{Sparse1}$	$t_{[61]}$	*median*	*max*	*median*	*max*	$t_{[61]}/t_{Sparse1}$
1306	62500	595.306	351.016	1.5e-16	3.1e-14	1.2e-14	1.8e-6	0.590
235	3140	63.888	39.275	1.5e-16	2.2e-16	1.5e-15	2.5e-10	0.615
2814	8256	6.752	5.232	1.5e-16	1.7e-14	1.2e-12	2.2e-7	0.775
1414	49702	11.746	9.138	1.5e-16	1.0e-11	5.9e-15	1.9e-8	0.778
1419	682862	305.073	329.030	4.0e-16	2.0e-7	4.5e-14	3.2e-5	1.079
35	2003	0.071	5.294	1.5e-16	2.2e-16	4.0e-15	1.9e-11	74.796
1912	13681	0.588	57.240	1.5e-16	2.2e-16	1.3e-14	9.0e-11	97.354
2540	9774	1.116	182.139	1.5e-16	2.2e-16	2.4e-15	1.5e-9	163.198
45	3134	0.074	15.231	1.5e-16	2.5e-16	2.0e-15	2.7e-10	205.098
1606	5489	0.245	149.531*	1.5e-16	2.2e-16	1.5e-11	1.4e-6	610.043*

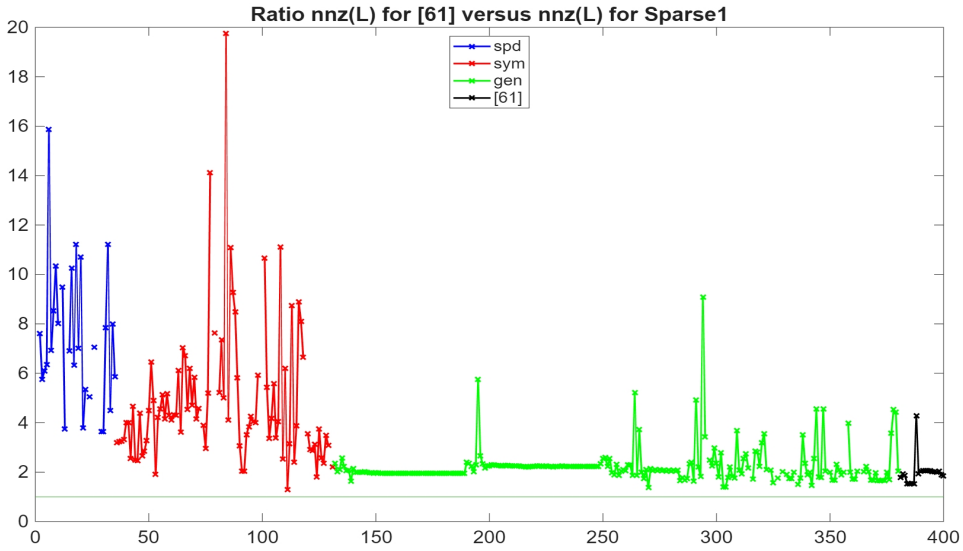

FIG. 5. *Ratio of nonzero elements of L-factor of [61] versus Sparse1*

proximation by Matlab's "backslash" and of the verified bounds computed by Sparse1 and [61]. The relative error of Sparse1 is mostly not far from maximum accuracy so that we can use the bounds to compute the relative error of the approximation by "backslash". As can be seen "backslash" computes usually approximations with some 14 to 15 correct figures, but sometimes only few figures are correct. In the median the inclusions by [61] are accurate to about 14 correct figures as well but that is mathematically certified. However, in some examples inclusion intervals are wide containing zero as an inner point.

Despite less fill-in of the L-factor, a main reason why Sparse1 is fast is that we try to minimize the effort to compute verified bounds. We discuss some details of our Algorithm `verifySparselss1` in Table 7 on the several improvement steps in the subalgorithms "verifySparseSPD1", "verifySparseNearSym1" and "verifySparseGen1".

We start with "verifySparseSPD1" which is called when the input matrix is symmetric. If this subalgorithm fails, then "verifySparseNearSym1" is called. If "veri-

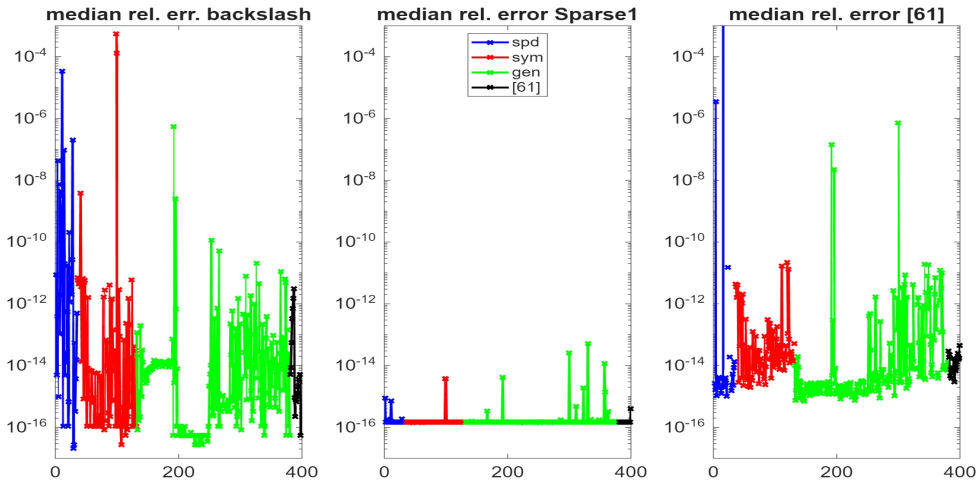

FIG. 6. *Median of relative errors of Matlab's "backslash", by Sparse1 and [61]*

TABLE 13
Detailed analysis of "verifySparseSPD1' for the 35 *spd test cases satisfying* (12.1)

action	step	number of times
decrease s	12	3
failure, call verifySparseNearSym1	13	0
improve α first time	16	3
improve α second time	17	0
verification failed, call verifySparseGen1	18	0

fySparseSPD1" fails in Step 17 with $\alpha \geqslant s$, then `verifySparselss1` calls "verifySparseGen1". That was not the case in all 35 examples in spd in Table 10. More details are displayed in Table 13. The algorithm is very stable. If the problem is too ill-conditioned, the initial value of s may be too large (which happened 3 times), and/or α is too large and has to be improved. The first improvement in Step 16 was necessary in 3 out of the 35 test cases, the second improvement never occurred.

Next we discuss subalgorithm "verifySparseNearSym1", details are in Table 14. From Matlab Version 2026a on the problem with singular D in Step 4 is solved, in previous releases it occurred occasionally. In 1 out of the 96 samples the anticipated lower bound s for $\sigma_{\min}(A)$ was so small that there was no hope to compute an inclusion successfully. In this case "verifySparseGen1" was called in Step 9 and computed an inclusion successfully. The improvement of α and β in steps 11 and 13 was used in 4 times each in the 96 tests in sym, i.e., in the remaining 92 cases the a priori bound (2.10) was sufficient. The improvement of α and β in Step 14 was necessary twice in 96 tests. As this is time consuming the numbers are in bold-face. Failure in Line 15 would also be time consuming but did not occur. Otherwise in few cases an improvement of γ was necessary, see Table 14.

Thirdly, some details on the performance of subalgorithm "verifySparseGen1" for the 249 "gen" test cases plus the 20 tests from [61]. Now we are treating the augmented matrix (9.1) so that more improvements of the used constants are necessary. The improvement of α and β in steps 12 and 14 was used in 16% and 24% of the tests,

TABLE 14
Detailed analysis of "verifySparseNearSym1' for the 96 *sym test cases satisfying* (12.1)

action	step	number of times
s too small	9	1
improve α	11	4
improve β	13	4
improve M, α and β	**14**	**2**
verification failed, call verifySparseGen1	15	0
decrease s	19	8
verification failed, call verifySparseGen	21	0
improve γ first time	23	4
improve γ second time	24	1
verification failed, call verifySparseGen1	25	0

respectively. The time consuming recomputation of M, α and β in Step 15 occurred in 3 out of the 269 test cases, and in 1 case, namely number 934 in [10], the verification finally failed in Step 16. The reasons will be explained later. The shift s for the Cholesky decomposition in Line 20 was decreased 11 times out of the 269 tests, and γ was improved 36 times out of the 269 tests in Step 24 of "verifySparseGen1", and improved again 2 times in Step 25.

TABLE 15
Detailed analysis of "verifySparseGen1' for the gen test cases satisfying (12.1) *and the 20 test cases in [61]*

action	step	number of times
improve α	12	43
improve β	14	65
improve M, α and β	**15**	**3**
verification failed, call verifySparseGen1	16	1
decrease s	20	11
improve γ first time	24	36
improve γ second time	25	2

TABLE 16
Timing and accuracy for sparse linear systems in [10] satisfying the conditions in (12.1) $\left[\ \varrho := t_{[61]}/t_{Sparse1}\ \right]$

	matrix				*times [sec]*			*relerr backslash*		*relerr Sparse*1		*relerr* [61]		
	# matrix	n	$nnz(A)$	$\kappa_2(A)$	$t_{backslash}$	$t_{Sparse1}$	$t_{[61]}$	*median*	*max*	*median*	*max*	*median*	*max*	ϱ
spd	358	1050	26198	4.6e15	0.00	0.13	1.53	8.8e-12	0.691	9.0e-16	5.2e-13	NaN	NaN	
	430	1733	22189	7.0e12	0.01	0.03	0.44*	4.2e-8	1.4e-4	1.8e-16	5.2e-13	3.5e-6	4.0e-4	15.32*
	411	2568	75628	1.2e15	0.00	0.09	8.79	3.4e-5	0.293	7.3e-16	1.3e-9	NaN	NaN	
	440	3363	99471	1.2e13	0.00	0.08	5.91	9.3e-8	5.4e-5	1.5e-16	5.4e-13	NaN	NaN	
	46	3562	159910	2.0e11	0.03	0.12	8.84*	2.0e-11	1.1e-6	1.5e-16	2.2e-16	1.8e206	1.8e206	74.67*
	1611	5357	207123	2.4e10	0.05	0.18	9.35	2.1e-10	7.9e-5	1.5e-16	4.5e-14	NaN	NaN	
	1606	5489	263351	1.8e8	0.02	0.25	149.53*	1.2e-12	8.1e-7	1.5e-16	2.2e-16	1.5e-11	1.4e-6	610.04*
	1607	5489	262943	1.8e10	0.02	0.24	25.39	1.5e-11	4.6e-6	1.5e-16	2.2e-14	NaN	NaN	
	1610	5489	217669	2.5e10	0.02	0.24	14.32	2.7e-11	1.8e-6	1.5e-16	1.3e-14	NaN	NaN	
	413	6867	98671	8.6e12	0.07	0.16	3.04	2.0e-7	0.001	1.9e-16	1.4e-9	NaN	NaN	
	47	15439	252241	4.4e12	0.04	0.41	17.45	4.9e-13	3.8e-7	1.5e-16	4.0e-16	NaN	NaN	
sym	2684	5578	41940	6.6e15	**0.14**	0.07	0.39	4.6e-15	8.8e-8	1.5e-16	2.2e-16	NaN	NaN	
	2412	8034	23626	4.5e12	0.03	0.72	0.60	1.7e-12	2.5e-4	1.5e-16	4.0e-15	NaN	NaN	
	2410	9000	26556	4.8e12	0.04	1.38	0.85	2.8e-12	2.4e-4	1.5e-16	8.2e-16	NaN	NaN	
	1210	20360	509866	6.1e14	0.22	488.36	174.02	5.5e-4	1.000	3.7e-15	2.4e-10	NaN	NaN	
	1451	20360	509866	6.1e14	0.26	492.10	174.29	1.3e-4	0.402	3.8e-15	6.6e-11	NaN	NaN	
	2536	43887	426898	5.9e14	0.06	3.41	4.21	2.4e-13	2.6e-4	1.5e-16	1.2e-12	6.9e-15	2.8e-9	1.24
	1304	50000	349968	2.4e8	**335.49**	9.15	162.79	1.5e-12	4.8e-7	1.5e-16	2.2e-16	NaN	NaN	
	1306	62500	424966	4.2e14	**1309.29**	595.31	351.02	6.0e-12	1.8e-4	1.5e-16	3.1e-14	1.2e-14	1.8e-6	0.59
	1300	80595	438795	1.2e8	1.05	9.34	89.97	3.1e-14	4.5e-8	1.5e-16	2.2e-16	NaN	NaN	
gen	1896	1020	5883	2.8e9	0.02	0.07	0.09	1.2e-13	7.0e-9	1.5e-16	3.8e-16	1.0e-14	3.6e-9	1.29
	243	1080	23094	9.6e11	0.01	0.31	0.37	1.7e-16	3.1e-11	1.5e-16	2.2e-16	2.7e-15	1.8e-10	1.21
	1072	1220	5892	4.8e12	0.01	0.04	0.06	9.3e-15	7.1e-5	1.5e-16	1.1e-15	1.1e-15	5.6e-13	1.46
	214	1374	8588	3.8e14	0.01	0.06	0.13	5.5e-7	0.038	4.1e-15	4.8e-8	1.4e-7	0.144	2.24
	438	1633	46626	8.3e10	0.01	0.41	1.60	2.5e-9	1.1e-5	1.5e-16	4.9e-16	NaN	NaN	
	893	1650	7419	5.6e12	0.01	0.03	0.07	6.9e-13	2.5e-6	1.5e-16	4.7e-12	2.2e-8	0.709	2.64
	1865	2053	18447	5.4e15	0.04	0.15	0.38*	4.4e-16	1.0e-5	1.5e-16	3.4e-14	4.8e-13	6.8e-10	2.51*

TABLE 17
Timing and accuracy for sparse linear systems in [10] satisfying the conditions in (12.1) $\left[\ \varrho := t_{[61]}/t_{Sparse1}\ \right]$

matrix				*times* [*sec*]			*relerr backslash*		*relerr Sparse*1		*relerr* [61]		
# matrix	*n*	*nnz*(*A*)	$\kappa_2(A)$	$t_{backslash}$	$t_{Sparse1}$	$t_{[61]}$	*median*	*max*	*median*	*max*	*median*	*max*	ϱ
546	2880	18229	3.5e13	0.02	0.29	0.45	7.4e-13	4.2e-7	1.5e-16	2.2e-16	1.7e-12	1.0e-6	1.54
465	2904	58142	1.4e12	0.02	0.38	0.43	1.1e-16	1.8e-10	1.5e-16	2.2e-16	5.8e-15	1.1e-8	1.12
235	3140	543160	4.8e9	0.73	63.89	39.28	1.1e-16	1.2e-11	1.5e-16	2.2e-16	1.5e-15	2.5e-10	0.61
737	4101	82682	4.1e12	0.15	0.63	1.10	1.0e-15	7.4e-5	1.5e-16	1.3e-10	2.7e-13	0.013	1.75
157	4929	33044	1.0e8	**0.29**	0.14	0.37*	1.0e-14	1.7e-10	1.8e-16	3.1e-5	5.4e-14	1.1e-10	2.75*
414	5773	71701	2.1e12	0.05	0.84	3.78	4.4e-16	2.3e-5	1.5e-16	3.3e-15	NaN	NaN	
818	6316	167178	3.9e14	0.01	1.19	1.47	1.1e-16	2.7e-15	1.5e-16	2.2e-16	1.1e-15	2.2e-12	1.24
934	7055	30082	2.8e13	0.02	1.20	0.69	3.3e-16	0.003	NaN	NaN	NaN	NaN	
739	7337	156508	2.5e13	0.63	1.32	2.25	8.9e-16	2.3e-5	1.5e-16	4.2e-10	1.6e-12	0.019	1.70
1395	7548	834222	4.8e12	2.90	17.64	126.97*	2.3e-13	3.6e-6	2.6e-14	9.9e-9	7.2e-7	0.007	7.20*
2814	8256	109368	2.1e15	0.89	6.75	5.23	4.7e-13	3.1e-5	1.5e-16	1.7e-14	1.2e-12	2.2e-7	0.77
580	9129	52883	3.9e13	0.10	2.23	2.95	2.7e-14	5.9e-9	1.5e-16	2.2e-16	9.5e-13	1.4e-7	1.32
581	9129	52883	8.4e13	0.09	2.28	3.09	4.4e-15	1.6e-7	1.5e-16	2.2e-16	7.8e-14	5.1e-6	1.36
741	10672	232633	7.6e13	1.52	2.16	3.20	7.8e-16	3.2e-5	1.5e-16	5.8e-8	4.2e-12	0.044	1.49
743	10964	233741	4.1e14	1.75	3.57	4.38	4.3e-13	0.002	4.7e-16	4.8e-5	8.6e-12	0.369	1.23
415	12005	259577	2.2e12	0.12	3.29	12.52	8.3e-16	3.3e-5	1.5e-16	1.5e-14	NaN	NaN	
2242	14098	252446	5.0e7	**7.46**	3.07	3.84	8.2e-15	5.8e-9	1.5e-16	2.2e-16	9.9e-15	2.7e-8	1.25
745	14270	307858	4.2e14	3.88	5.01	4.30	9.9e-13	0.001	1.9e-15	7.6e-4	NaN	NaN	
2233	15374	610299	9.5e7	9.43	20.02	23.48	8.8e-15	4.4e-9	1.5e-16	2.2e-16	1.4e-14	1.5e-10	1.17
756	15606	61484	5.9e10	0.27	0.94	1.03	2.0e-11	3.2e-5	1.5e-16	1.4e-14	NaN	NaN	
922	16428	948696	2.6e13	3.78	329.76	920.64	1.7e-13	1.2e-7	1.5e-16	7.1e-14	NaN	NaN	
747	17576	381975	4.0e14	6.56	113.15	7.88	4.5e-12	0.005	5.2e-14	0.004	NaN	NaN	
582	18289	106803	6.8e13	0.21	6.85	9.94	3.1e-14	8.4e-9	1.5e-16	3.6e-16	1.6e-12	1.1e-6	1.45
583	18289	106803	5.5e13	0.20	7.21	9.63	1.6e-15	1.4e-7	1.5e-16	6.4e-14	5.6e-14	1.8e-5	1.34
431	19716	227872	2.2e12	0.13	4.35	6.11	4.2e-16	2.0e-4	1.8e-16	7.1e-10	NaN	NaN	
2234	20374	812749	9.5e7	19.03	24.15	35.98	9.4e-15	8.8e-8	1.5e-16	2.2e-16	1.4e-14	2.4e-9	1.49
1109	25187	193276	2.2e12	**12.48**	2.68	3.54	3.9e-13	7.0e-7	1.5e-16	2.8e-16	1.9e-11	1.8e-5	1.32

TABLE 18
Timing and accuracy for sparse linear systems in [10] satisfying the conditions in (12.1) $\left[\ \varrho := t_{[61]}/t_{Sparse1}\ \right]$

matrix				*times [sec]*			*relerr backslash*		*relerr Sparse1*		*relerr* [61]		
# *matrix*	*n*	*nnz*(*A*)	$\kappa_2(A)$	$t_{backslash}$	$t_{Sparse1}$	$t_{[61]}$	*median*	*max*	*median*	*max*	*median*	*max*	ϱ
584	27449	160723	1.1e14	0.31	12.24	18.48	3.7e-14	1.5e-8	1.5e-16	1.0e-14	2.3e-12	4.9e-6	1.51
585	27449	160723	5.5e13	0.32	11.97	18.01	1.3e-15	1.2e-6	1.5e-16	8.1e-14	9.0e-14	7.3e-5	1.50
574	27534	151063	2.3e14	0.93	4.59	14.23	5.6e-16	3.9e-7	1.6e-16	2.9e-9	8.1e-12	0.245	3.10
576	34454	190224	3.8e14	1.21	6.47	18.93	4.4e-16	4.2e-6	1.8e-16	7.7e-6	1.9e-11	0.906	2.93
586	36609	214643	2.7e14	0.47	18.01	27.21	5.5e-14	4.0e-8	1.5e-16	3.0e-12	3.3e-12	6.0e-4	1.51
587	36609	214643	5.6e13	0.42	17.50	26.72	2.1e-15	2.2e-7	1.5e-16	2.3e-9	4.8e-14	0.508	1.53
2240	38098	684206	5.0e7	**127.79**	10.33	11.89	7.5e-15	2.3e-9	1.5e-16	2.2e-16	1.1e-14	1.2e-8	1.15
578	41374	229385	6.7e14	1.43	10.88	23.31	4.4e-16	2.0e-6	1.2e-14	0.019	NaN	NaN	
579	41374	229385	1.1e7	0.78	10.39	70.87*	6.4e-15	8.6e-10	1.4e-15	1.1e-6	6.6e-14	3.3e-9	6.82*
588	45769	268563	7.1e14	1.14	25.86	36.54	5.8e-14	1.2e-6	1.5e-16	8.9e-9	7.1e-12	0.690	1.41
589	45769	268563	5.6e13	0.53	23.46	36.39	1.9e-15	4.3e-7	1.5e-16	7.9e-7	1.4e-13	17.735	1.55
1413	49702	333029	9.0e14	**106.16**	18.54	8.88	3.3e-16	0.006	3.3e-16	1.3e-9	NaN	NaN	
1414	49702	332807	4.0e13	**102.84**	11.75	9.14	7.7e-15	9.3e-6	1.5e-16	1.0e-11	5.9e-15	1.9e-8	0.78
983	51993	380415	5.9e15	77.47	120.86	487.78	1.1e-11	3.7e-6	1.5e-16	2.2e-16	NaN	NaN	
590	54929	322483	1.9e15	1.34	31.75	46.99	7.7e-14	2.7e-7	1.5e-16	6.9e-7	1.2e-11	45.802	1.48
591	54929	322483	5.6e13	0.63	31.52	45.62	1.6e-15	1.2e-5	1.5e-16	1.3e-4	1.2e-13	3440.240	1.45
592	64089	376395	5.0e15	1.55	37.91	57.55	8.4e-14	4.5e-6	1.5e-16	2.1e-4	9.9e-12	2.0e4	1.52
593	64089	376395	5.6e13	0.74	38.21	57.79	1.7e-15	3.2e-6	1.5e-16	0.021	1.2e-13	7.6e5	1.51
373	80209	307604	2.8e11	0.75	3.63	17.81	4.8e-12	1.6e-5	1.6e-16	5.9e-10	9.8e-15	8.6e-5	4.91
2657	87936	593276	6.7e8	**1850.68**	7.14	10.47	1.4e-13	7.9e-8	1.5e-16	1.3e-15	9.0e-15	2.2e-9	1.47
1344	94294	479246	5.6e12	**82.65**	5.99	41.94	1.4e-14	1.1e-6	1.5e-16	2.4e-10	9.1e-15	4.2e-7	7.00
[61] 918	11532	44206	3.5e12	**1.00**	0.76	1.16	8.6e-15	6.8e-10	1.5e-16	2.2e-16	1.8e-14	3.8e-10	1.54
919	16428	63406	2.2e13	**2.80**	1.35	2.04	5.5e-14	2.5e-9	1.5e-16	2.8e-16	2.2e-14	3.0e-9	1.51
2567	40948	412148	2.8e9	0.15	2.70	3.87	3.1e-12	1.0e-7	1.5e-16	3.9e-16	9.0e-15	3.4e-10	1.44
1417	321821	1931828	5.1e22	**memory**	17.28	49.13			1.5e-16	1.0e-13	1.8e-14	7.7e-8	2.84
1419	682862	2638997	8.1e19	**crash**	305.07	329.03			4.0e-16	2.0e-7	4.5e-14	3.2e-5	1.08

To show all data is too much for this note, so we put the results for all 400 test cases into [66]. We present some selected data in Tables 16 - 18. Here NaN in the relative error columns indicates failure of verification, and otherwise, the columns are self-explaining. The computing time of Matlab's "backslash" is printed in bold-face if slower than Sparse1. The median and maximum relative error of the approximation by "backslash" is computed by the error bounds provided by Sparse1. For the example number 934 in [10] where Sparse1 (and [61]) failed, we used the verified and maximally accurate inclusion by Algorithm Sparse2 to be introduced in Part II of this note which did not fail in all our test cases. The ratio of computing times $t_{[61]}/t_{Sparse}$ is only displayed when [61] ended successfully.

In order to reduce space for the results to be displayed to 3 pages, we considered the 400 examples in (12.1) according to Table 19 including the 20 tests in [61]. That

TABLE 19
Displayed tests extracted from the 400 *tests in Table 10*

- [61] failed with `precond`=1 and was recomputed with `precond`=0
- Sparse1 failed with `precond`=1 and was recomputed with `precond`=0
- all tests where one of "backslash", Sparse1 or [61] failed
- all tests where the median relative error by Sparse1 is larger than 10^{-15}
- all tests where the maximal relative error by Sparse1 is larger than 10^{-10}
- all tests where the median relative error by [61] is larger than 10^{-2}
- all tests where the maximal relative error by [61] is larger than 10^{-2}
- all tests where the computing time ratio $t_{[61]}/t_{Sparse}$ is less than 1.54

means in particular that if a test is *not* listed here in Tables 16 - 18 but only in [66], then both Sparse1 and [61] succeeded, Sparse1 guarantees at least 10 correct digits of all entries of the inclusion, and Sparse1 is at least 1.54 times faster than [61]. The curios ratio 1.54 of computing time $t_{[61]}/t_{Sparse1}$ is tuned to fill 3 pages of results.

As has been mentioned in Section 11, for the symmetric positive definite matrix number 46 in [10] inclusion failed with equilibration, succeeded without in [61] but the residual iteration diverged resulting in identical inclusion intervals $[-10^{206}, +10^{206}]$ for all entries.

In two cases we observed failure of Matlab's "backslash". In example 1417 from [10] 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. As a consequence, the columns referring to the relative error of "backslash" remain empty.

Numerical evidence suggests that Algorithm `verifySparselss1` succeeds to compute verified error bounds for 2-norm condition numbers close to $\mathbf{u}^{-1}$. The only failure, namely matrix number 934 in [10] with `condest(A)`$= 1.7 \cdot 10^{12}$ has 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 usually producing a reliable estimate, and that is confirmed using the multiple precision package [18]. However, `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 $2.8 \cdot 10^{13}$.

We give some additional test results for randomly generated ill-conditioned sparse matrices as in (12.2). The resulting matrices have some 100,000 nonzero elements each, and the median condition number over the 100 tests was $1.9 \cdot 10^{15}$.

Sometimes generally valid rules of thumb 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 [60, 25] 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 20 which might not be typical for sparse systems. The median 2-norm condition number $1.9 \cdot 10^{15}$ of our

TABLE 20
Results for 100 *randomly generated ill-conditioned test cases*

	Sparse1	[61]
inclusions	failed in 4 out of 100 tests	failed in 38 out of 100 tests
median relative error	1.5e-16	1.5e-14
maximal relative error	1.2e-8	170.252

samples is boarder line in the sense that a verification algorithm might just succeed to compute verified bounds. Still, Sparse1 succeeds in 96 out of the 100 cases to compute bounds with at least 8 coinciding figures in each entry. In the median inclusions are maximally accurate.

The algorithm in [61] succeeds in 62 out of 100 cases, however, in 27 out of the 62 successful cases the relative error of some entries exceeded 0.01, in 18 cases it exceeds 0.5, i.e., barely one figure can be guaranteed.

For the randomly generated ill-conditioned matrices the algorithm in [61] is in the median 1.1 times slower and at most 1.1 times faster than Sparse1. Conversely, Sparse1 is up to 3.1 times faster than [61] and fails in significantly less cases.

In 35 out of the 100 random test cases Matlab's "backslash" produced an approximation with relative error exceeding 0.01 for some entries, and in the median Sparse1 is 5.7 times slower than "backslash". The complete set of results can be found in [66]. Tests for complex data are postponed to Part II of this note because [61] handles only real data.

We close this section with two benchmark problems. The first is an ill-conditioned synthetic saddle-point system

$$K = \begin{pmatrix} A & B \\ B^T & \varepsilon I_m \end{pmatrix},$$

a representative of constrained PDE systems such as Stokes flow, mixed finite-element discretizations, or KKT systems in optimization. The velocity block $A \in \mathbb{R}^{n \times n}$ is chosen as a scaled identity matrix $A = \nu I_n$ with small diffusion parameter $\nu = 10^{-14}$, resulting in a cluster of small eigenvalues. Hence K is symmetric indefinite and nearly singular.

The constraint matrix $B \in \mathbb{R}^{n \times m}$ is generated as a random sparse matrix with one nonzero entry per column, thereby enforcing $m = 25,000$ independent linear constraints on the $n = 50{,}000$ velocity unknowns. The coupling in the lower-left block of K by B^T is characteristic of mixed formulations and Karush–Kuhn–Tucker systems.

To avoid exact singularity while preserving the saddle-point structure, a vanishingly small Tikhonov regularization $\epsilon = 10^{-16}$ is added to the lower right block of K. This results in an indefinite matrix with eigenvalues spanning many orders of magnitude. The combination of extremely small diffusion, nearly rank-deficient constraint coupling, and minimal regularization leads to an ill-conditioned linear system. The right hand side is chosen randomly of norm 1.

The results are shown as "problem 1" in Table 21. The condition number of the original matrix is $8 \cdot 10^{16}$. Matlab's backslash operator is very fast, however many entries are poor approximations. While 87% of the entries have 15 correct figures, the remaining 13% are wrong by at least a factor 2, and in 104 of the 75,000 entries the sign is incorrect. However, these entries are small compared to $\|K^{-1}b\|_\infty = 10^{14}$.

The method in [61] needs about the same computing time as Sparse1, however, the inclusions are of less quality. Some 40% of the inclusions are nearly maximally accurate, while for the remaining entries not a single digit can be guaranteed. However, those entries are small compared to $\|K^{-1}b\|_\infty = 10^{14}$. But 13% of the inclusions are equal to $[-6.6 \cdot 10^{15}, 6.6 \cdot 10^{15}]$. As discussed previously the reason is lack of convergence due to the factorization of the shifted instead of the original matrix.

We tried iterative methods. Here `gmres` delivered some 6 to 8 correct digits in less than a second, `bcgstab` required only 0.08 seconds for 2 to 4 correct digits, and computing a preconditioner by `ilu` failed.

Our method Sparse1 is maximally accurate in the median, and at least 8 figures of all entries are guaranteed. As by Algorithm `verifylssSparse1` we first try "verifySparseSPD1". The Cholesky decomposition in Step 5 fails quickly and "verifySparseNearSym1" is called. Here, however, equilibration in Step 2 is extremely counterproductive by increasing the condition number from $8 \cdot 10^{16}$ to $1.2 \cdot 10^{29}$. Thus "verifySparseGen1" is called and computes the inclusion. In contrast, for that general case the unsymmetric equilibration (3.3) reduces the condition number to 19.2.

We can force an unsymmetric equilibration by changing the number of iterations in (3.2) to 3 or 5. That reduces the condition number from $8 \cdot 10^{16}$ to $2.5 \cdot 10^{15}$. Now "verifySparseNearSym1" succeeds to compute an inclusion in about half the computing time, and all inclusions are maximally accurate.

TABLE 21
Results for two benchmark sparse linear systems

	problem 1			problem 2		
	backslash	[61]	Sparse1	backslash	[61]	Sparse1
n		75,000			60,000	
nnz(A)		125,000			357,594	
condest(A)		$8.0 \cdot 10^{16}$			$4.7 \cdot 10^{14}$	
time [sec]	0.056	2.45	2.75	> 24 hours	ϱ failed	4.5
median rel. error	0	0.72	$1.5 \cdot 10^{-16}$			$1.6 \cdot 10^{-16}$
wrong sign	104	9957	0			0
max rel. error	1	$6.5 \cdot 10^{15}$	$1.0 \cdot 10^{-8}$			$2.2 \cdot 10^{-16}$

The second problem a saddle-point system arising from a finite-difference discretization of the two-dimensional incompressible Stokes equations,

$$-\nu\Delta\mathbf{u} + \nabla p = \mathbf{f}, \qquad \nabla \cdot \mathbf{u} = 0,$$

on a rectangular, anisotropic grid with $(nx, ny) = (100, 200)$. The anisotropy ($\Delta x > \Delta y$) leads to a stronger numerical stiffness in the fine-grid direction. The velocity

components (u, v) and the pressure p are placed at cell centers.

The diffusion operators for the velocity field are assembled using standard second-order central finite differences. The incompressibility constraint is enforced by the discrete divergence operators D_x and D_y, constructed from centred differencing such that D_x^T and D_y^T represent the corresponding discrete gradient operators.

This yields the symmetric saddle-point system

$$K = \begin{pmatrix} A & 0 & D_x^T \\ 0 & A & D_y^T \\ D_x & D_y & 0 \end{pmatrix},$$

which is characteristic of mixed formulations of incompressible flow. To remove the nullspace associated with the constant pressure mode, a single Dirichlet reference condition is imposed by fixing one degree of freedom. The right-hand side vector f is chosen as a synthetic load.

The results are shown as "problem 2" in Table 21. The iterative solvers `gmres` and `bcgstab` did not converge, and computing a preconditioner by `ilu` failed. We stopped the backslash operator after 24 hours of computing time.

The algorithm in [61] failed as well because the call in Step 4 of Table 8 needed 142 seconds to try to compute an approximation of ϱ, but finally fails.

Our solver Sparse1 required 4.5 seconds to compute maximally accurate inclusions of all entries of the solution.

13. Conclusion and an open problem. We presented an algorithm 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 "backslash" 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 almost all examples out of the Suite Sparse Matrix Collection [10] our Algorithm `verifySparselss1` succeeds to compute accurate error bounds, often with close to maximal accuracy, i.e., all bounds differ by few bits. Our algorithm seems promising for the solution of larger verification problems, in particular for partial differential equations.

The methods in Part I and also in Part II of this note are based on a matrix decomposition. There are numerous iterative methods including Krylov or stationary iterative methods to compute an approximate solution, often an attractive way to attack sparse systems. There are error estimates, but those are qualitative and/or theoretical and not computable. Up to now some factorization is the only way for the step from a small residual to a verified inclusion. The use of iterative methods as the basis of verification is a completely open problem.

Acknowledgement. Many thanks to Takeshi Terao and Katsuhisa Ozaki for reading an earlier version of this manuscript and useful comments. My special thanks go to the reviewers for their extremely valuable and constructive comments. They helped to improve the paper significantly.

REFERENCES

[1] F.L. Bauer. Optimally scaled matrices. *Numerische Mathematik 5*, pages 73–87, 1963.
[2] M. Benzi, G.H. Golub, and J. Liesen, "Numerical solution of saddle point problems," *Acta Numerica*, vol. 14, pp. 1–137, 2005.
[3] 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.
[4] 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.
[5] F. Bünger. Shrink wrapping for Taylor models revisited. *Numerical Algorithms*, 78(4):1001–1017, 2018.
[6] F. Bünger. Preconditioning of Taylor models, implementation and test cases. *Nonlinear Theory and its Applications (NOLTA)*, 12(1):2–40, 2021.
[7] P.A. Businger. Matrices which can be optimally scaled. *Numer. Mathematik*, 12:346–348, 1968.
[8] Erin Carson and Nicholas J. Higham. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM Journal on Scientific Computing*, 39(6):A2834–A2856, 2017.
[9] L. Collatz. Einschließungssatz für die charakteristischen Zahlen von Matrizen. *Math. Z.*, 48:221–226, 1942.
[10] T.A. Davis, Y. Hu: The University of Florida Sparse Matrix Collection. ACM Transactions on Mathematical Software 38, 1, Article 1, 2011.
[11] 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.
[12] J. Demmel, Y. Hida. Accurate and efficient floating point summation. *SIAM J. Sci. Comput. (SISC)*, 25:1214–1248, 2003.
[13] 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.
[14] 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.
[15] H.C. Elman, D.J. Silvester, and A.J. Wathen, *Finite elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics*, 2nd ed., Oxford University Press, 2014.
[16] G.E. Forsythe and E.G. Straus. On best conditioned matrices. *Proc. Amer. Math. Soc. 6*, pages 340–355, 1958.
[17] N. J. Higham: *Accuracy and Stability of Numerical Algorithms*, SIAM Publications, Philadelphia, 2nd edition, 2002.
[18] P. Holoborodko. *Multiprecision Computing Toolbox for MATLAB*. Advanpix LLC., Yokohama, Japan, 2026.
[19] R. A. Horn, C. R. Johnson: *Matrix Analysis*, second edition. Cambridge University Press, 2013.
[20] R. A. Horn, C. R. Johnson: *Topics in Matrix Analysis*. Cambridge University Press, 1991.
[21] IEEE Standard for Floating-point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.
[22] 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.
[23] D. E. Knuth: *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison Wesley, Reading, Massachusetts, 1969.
[24] 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.
[25] F. R. de Hoog, R. S. Anderssen, M. A. Lukas: Differentiation of matrix functionals using triangular factorization. *Math. Comp.*, 80(275): 1585–1600, 2011.
[26] 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.
[27] P. Knight. The Sinkhorn–Knopp algorithm: Convergence and applications. *SIAM Journal on Matrix Analysis and Applications*, 30(1):261–275, 2008.
[28] 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.
[29] 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.
[30] M. Lange, S.M. Rump. Sharp estimates for perturbation errors in summations. *Math.Comp.*, 88:349–368, 2019.
[31] M. Lange. private communication. 2024.

[32] R. Lohner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*. PhD thesis, University of Karlsruhe, 1988.
[33] 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.
[34] M. Malcolm. On accurate floating-point summation. *Comm. ACM*, 14(11):731–736, 1971.
[35] MATLAB. User's Guide, Pre-Release 2026a, the MathWorks Inc., 2025.
[36] 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.
[37] 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.
[38] A. Neumaier. Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen. *Zeitschrift für Angew. Math. Mech. (ZAMM)*, 54:39–51, 1974.
[39] A. Neumaier: *Interval methods for systems of equations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1990.
[40] T. Ogita, S. M. Rump, S. Oishi: Accurate sum and dot product. *SIAM Journal on Scientific Computing (SISC)*, 26(6):1955–1988, 2005.
[41] T. Ogita, S.M. Rump, S. Oishi. Verified Solutions of Sparse Linear Systems by LU factorization, 2005.
[42] 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].
[43] K. Ozaki, T. Ogita, F. Bünger, S. Oishi. Accelerating interval matrix multiplication by mixed precision arithmetic. *Nonlinear Theory and its Applications IEICE*, 6(3):364–376, 2015.
[44] 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.
[45] S.M. Rump. *Kleine Fehlerschranken bei Matrixproblemen*. PhD thesis, Universität Karlsruhe, 1980.
[46] S.M. Rump. Approximate inverses of almost singular matrices still contain useful information. Technical Report 90.1, Forschungsschwerpunkt Informations- und Kommunikationstechnik, TU Hamburg-Harburg, 1990.
[47] 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.
[48] S.M. Rump. Verified computation of the solution of large sparse linear systems. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 75:S439–S442, 1995.
[49] S. M. Rump: INTLAB – INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Springer Netherlands, Dordrecht, 1999.
[50] 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.
[51] S.M. Rump. Inversion of extremely ill-conditioned matrices in floating-point. *Japan J. Indust. Appl. Math. (JJIAM)*, 26:249–277, 2009.
[52] S. M. Rump: Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010.
[53] S.M. Rump. Fast interval matrix multiplication. *Numerical Algorithms*, 61(1):1–34, 2012.
[54] 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.
[55] 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.
[56] S.M. Rump, T. Ogita. Super-fast validated solution of linear systems. *Journal of Computational and Applied Mathematics (JCAM)*, 199(2):199–206, 2007. Special issue on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN 2004).
[57] R. Skeel. Iterative refinement implies numerical stability for Gaussian elimination. *Math. Comp.*, 35(151):817–832, 1980.
[58] A. van der Sluis. Condition numbers and equilibration of matrices. *Numer. Mathematik*, 14:14–23, 1969.
[59] 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.
[60] G. W. Stewart: On the perturbation of LU, Cholesky, and QR factorizations. *SIAM J. Matrix*

Anal. Appl., 14: 1141–1146, 1993.
[61] Terao T., K. Ozaki. Method for verifying solutions of sparse linear systems with general coefficients. Applied Mathematics and Computation, Volume 490, April 2025, 129204.
[62] J.H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, 1965.
[63] T. Yamamoto. Error bounds for approximate solutions of systems of equations. *Japan J. Appl. Math.*, 1:157–171, 1984.
[64] 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.
[65] G. Zielke, V. Drygalla. Genaue Lösung linearer Gleichungssysteme. *GAMM Mitt. Ges. Angew. Math. Mech.*, 26:7–108, 2003.
[66] Supplementary material to *Verified error bounds for sparse systems Part I*, 2025.