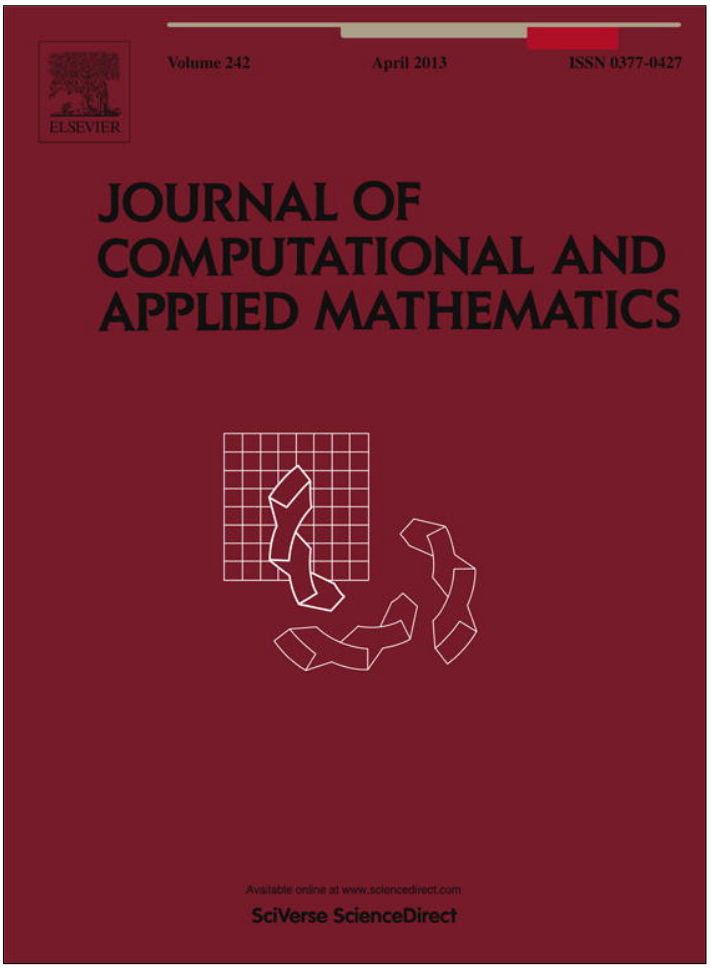

Journal of Computational and Applied Mathematics 242 (2013) 185–212

Contents lists available at SciVerse ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Accurate solution of dense linear systems, Part II: Algorithms using directed rounding

Siegfried M. Rump *

Institute for Reliable Computing, Hamburg University of Technology, Schwarzenbergstraße 95, Hamburg 21071, Germany
Waseda University, Faculty of Science and Engineering, 3–4–1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan

ARTICLE INFO

Article history:
Received 12 August 2011
Received in revised form 22 August 2012

MSC:
65F05
65G20

Keywords:
Linear systems
Matlab
Data with tolerances
Inner bounds
(Extremely) ill-conditioned matrices
Rigorous error bounds

ABSTRACT

In Part I and this Part II of our paper we investigate how extra-precise evaluation of dot products can be used to solve ill-conditioned linear systems rigorously and accurately. In Part I only rounding to nearest is used. In this Part II we improve the results significantly by permitting directed rounding. Linear systems with tolerances in the data are treated, and a comfortable way is described to compute error bounds for extremely ill-conditioned linear systems with condition numbers up to about $\mathbf{u}^{-2}/n$, where $\mathbf{u}$ denotes the relative rounding error unit in a given working precision. We improve a method by Hansen/Bliek/Rohn/Ning/Kearfott/Neumaier. Of the known methods by Krawczyk, Rump, Hansen et al., Ogita and Nguyen we show that our presented Algorithm `LssErrBnd` seems the best compromise between accuracy and speed. Moreover, for input data with tolerances, a new method to compute componentwise inner bounds is presented. For not too wide input data they demonstrate that the computed inclusions are often almost optimal. All algorithms are given in executable Matlab code and are available from my homepage.

1. Introduction and notation

The paper divides into two parts: In Part I all algorithms use only the four basic floating-point operations in rounding to nearest, in the present Part II we use directed rounding and methods different from Part I to obtain superior results. All algorithms in both parts are presented in executable Matlab code.

The methods in Part I are based on norm estimates, verifying convergence of some residual matrix by approximating its Perron vector. In this Part II we verify the H-property of some matrix and demonstrate how this can be used to effectively compute verified error bounds of the solution of a linear system. Moreover, we show an efficient method to compute so-called "inner" inclusions of a linear system, the data of which is afflicted with tolerances.

Dividing the paper into two parts also serves didactical purposes. In Part I we demonstrate that using only rounding to nearest allows us to give simple algorithms to produce rigorous results. The algorithms presented in Part II can still be formulated in rounding to nearest, however, at the cost of easy readability.

All algorithms are given in executable Matlab code for which we reserve the "verbatim"-font. For instance, $\mathtt{C} = \mathtt{A} * \mathtt{B}$ means that $\mathtt{C}$ is the result of the floating-point multiplication $\mathtt{A} * \mathtt{B}$, where $\mathtt{A}$ and $\mathtt{B}$ are compatible quantities (scalar, vector, matrix). For analyzing the error we use ordinary mathematical notation, for example in $P = \mathtt{A} \cdot \mathtt{B}$ the verbatim-font is used for floating-point quantities so that P is the exact (real) product of $\mathtt{A}$ and $\mathtt{B}$. For $\mathtt{A}, \mathtt{B} \in \mathbb{F}^{n\times n}$ this implies $|P - \mathtt{C}| \sim \mathbf{u}|\mathtt{A}| \cdot |\mathtt{B}|$.

* Correspondence to: Institute for Reliable Computing, Hamburg University of Technology, Schwarzenbergstraße 95, Hamburg 21071, Germany.
E-mail address: rump@tu-harburg.de.

0377-0427/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2012.09.024

Comparison between vectors and matrices is always to be understood entrywise, for example $x \le y$ for $x, y \in \mathbb{R}^n$ means $x_i \le y_i$ for $1 \le i \le n$. For $A \in \mathbb{R}^{n\times n}$, Ostrowski's comparison matrix $\langle A\rangle \in \mathbb{R}^{n\times n}$ is defined by

$$\langle A\rangle_{ii} := |A_{ii}| \quad \text{and} \quad \langle A\rangle_{ij} := -|A_{ij}| \text{ for } i \ne j. \tag{1.1}$$

If $\langle A\rangle$ is an M-matrix, then A is called an H-matrix. In that case A and $\langle A\rangle$ are non-singular, and $|A^{-1}| \le \langle A\rangle^{-1}$. Finally, $\varrho(A)$ denotes the spectral radius of A.

This Part II of the paper is organized as follows. In the next section we discuss several methods to obtain rigorous error bounds for linear systems based on H-matrices. In particular an efficient and best way is shown how to verify the H-property. In Section 3 we discuss how to implement these methods for rigorous error bounds using directed rounding.

Up to this point, standard Matlab suffices. From Section 4 on, the algorithms become too involved and we use INTLAB [1], the Matlab toolbox for reliable computing. The toolbox INTLAB is entirely written in Matlab and thus portable in many environments. The only features of INTLAB we need are the basic interval operations, so other libraries such as Intlib [2], Profil/Bias [3,4] or b4m [5] may be used as well.

In Section 4 linear systems, the data of which are afflicted with tolerances, are treated. Using interval operations the code becomes easier to read without sacrificing performance and/or accuracy. Now the floor is prepared to discuss alternative approaches to compute rigorous error bounds in Section 5. In particular a method originated by Hansen is discussed and improved. In Section 6 we show how to obtain inclusions for extremely ill-conditioned matrices, i.e. with condition number up to $\mathbf{u}^{-2}$, and finally we show a new method to calculate inner inclusions, even if only one entry of the matrix and/or the right hand side is afflicted with a tolerance. This allows us to judge the quality of outer inclusions. Detailed computational results and a conclusion finish the paper.

2. Rigorous error bounds for linear systems

Let a linear system $Ax = b$ with $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ be given. As in Part I let $R \in \mathbb{R}^{n\times n}$ be an approximate inverse of A, for example computed by the Matlab command `inv`. Note that there are no *a priori* assumptions on A and R, in particular no accuracy requirement on R. For the following explanation assume the matrices A and R to be non-singular; in the following theorems this will be verified *a posteriori* by the methods.

Define $C := RA$. For an approximation $\widetilde{x}$ to the solution $A^{-1}b$ we show several ways how to estimate

$$\Delta := \widetilde{x} - A^{-1}b = C^{-1}R(A\widetilde{x} - b) = C^{-1}c \quad \text{with } c := R(b - A\widetilde{x}). \tag{2.1}$$

In Part I of this paper we used normwise error estimates. Define $T := \mathrm{diag}(t)$ for a positive vector $t \in \mathbb{R}^n$. Defining $F := I - C$ and exploring

$$C^{-1}c = c + C^{-1}Fc = c + T(I - T^{-1}FT)^{-1}T^{-1}Fc \tag{2.2}$$

and using $|Ex| \le \|x\|_\infty \cdot |E|e$ for $E \in \mathbb{R}^{n\times n}$, $x \in \mathbb{R}^n$ and $e := (1, \ldots, 1) \in \mathbb{R}^n$ we obtain

$$\begin{aligned} |\widetilde{x} - A^{-1}b| &\le |c| + \|(I - T^{-1}FT)^{-1}T^{-1}Fc\|_\infty \cdot Te \\ &\le |c| + \frac{\|T^{-1}Fc\|_\infty}{1 - \|T^{-1}FT\|_\infty} \cdot t, \end{aligned} \tag{2.3}$$

provided $\|T^{-1}FT\|_\infty < 1$ (see Theorem 4.14 in Part I). Note that this is true for any $0 < t \in \mathbb{R}^n$. Since $\|T^{-1}FT\|_\infty = \|T^{-1}|F|T\|_\infty = \|T^{-1}|F|Te\|_\infty$, the obvious choice for Te is the Perron vector t of $|F|$. Then $|F|t = \varrho t$ implies $T^{-1}|F|Te = \varrho e$, minimizing $\|T^{-1}FT\|_\infty$. For very ill-conditioned matrices, the choice $t = e$ may fail due to $\|F\|_\infty \ge 1$, see Fig. 4.1 in Part I; otherwise, however, due to rounding errors in finite precision, sometimes the choice $t = e$ is superior to the Perron vector.

The previous result is based on the Neumann expansion $C^{-1} = (I - F)^{-1} = I + C^{-1}F$. If A is not too ill-conditioned, then $C = RA$ is not too far from the identity matrix and likely to be an H-matrix, which means that $\langle C\rangle$ is an M-matrix. The matrix C is an H-matrix, also called generalized diagonally dominant, if and only if there exists some positive $v \in \mathbb{R}^n$ such that $u := \langle C\rangle v > 0$. For the moment assume such a vector v to be given.

Then $C = RA$ is an H-matrix, so that A and R are non-singular. Denote by

$$\langle C\rangle := D - E \quad \text{with } D > 0, E \ge 0 \tag{2.4}$$

the splitting of $\langle C\rangle$ into diagonal and off-diagonal part. Note that $u = \langle C\rangle v > 0$ implies $D > 0$. Set

$$G := I - \langle C\rangle D^{-1} = ED^{-1} \ge 0. \tag{2.5}$$

Following [6] define

$$w_k := \max_i \frac{G_{ik}}{u_i} \quad \text{for } 1 \le k \le n, \tag{2.6}$$

so that $0 \le w \in \mathbb{R}^n$ and $G \le uw^T$. Multiplying $I - \langle C\rangle D^{-1} \le uw^T$ from the left by $\langle C\rangle^{-1} \ge 0$ and using $|C^{-1}| \le \langle C\rangle^{-1}$, which is true [7] for any matrix C, implies

$$|C^{-1}| \le \langle C\rangle^{-1} \le D^{-1} + vw^T. \tag{2.7}$$

Direct application of (2.7) to (2.1) gives

$$|\widetilde{x} - A^{-1}b| \le (D^{-1} + vw^T)|c|. \tag{2.8}$$

In contrast to the previous approach we use an implicit preconditioning of the matrix $\langle C\rangle$ to ensure $[D^{-1}\langle C\rangle]_{ii} \equiv 1$. For the splitting $C := \tilde{D} - \tilde{E}$ into diagonal and off-diagonal parts we have $|\tilde{D}| = D$ and $|\tilde{E}| = E$, so that the identity $C^{-1} = \tilde{D}^{-1}(I + \tilde{E}C^{-1})$, $|C^{-1}| \le \langle C\rangle^{-1}$ and $\widetilde{x} - A^{-1}b = C^{-1}c$ yield

$$|\widetilde{x} - A^{-1}b| \le \epsilon \Rightarrow |\widetilde{x} - A^{-1}b| \le D^{-1}\big(|c| + E\epsilon\big). \tag{2.9}$$

As in (2.2) we may apply $C^{-1} = (I - F)^{-1} = I + C^{-1}F$ to $\Delta = C^{-1}c$ to obtain

$$|\widetilde{x} - c - A^{-1}b| \le \big(D^{-1} + vw^T\big)|Fc|. \tag{2.10}$$

Finally expanding the Neumann series one more term gives $C^{-1} = I + F + C^{-1}F^2$, and therefore

$$|\widetilde{x} - c - Fc - A^{-1}b| \le \big(D^{-1} + vw^T\big)|F^2c|. \tag{2.11}$$

All the established bounds have to be estimated covering rounding errors which will be discussed in the next section. When relaxing the bounds (2.10) and (2.11) using $|Fc| \le |F| \;\|\; c|$, etc., the estimates become particularly simple because the products are just computed in rounding to upwards. Extensive tests suggest that this weakens the computed bounds marginally.

We compared the normwise bounds (2.3) choosing $e = (1, \ldots, 1)^T$ and the Perron vector of $|F|$ for t with (2.8), (2.9) applied to (2.8), (2.10) and (2.11) for various types of matrices with various right hand sides. We used in particular the matrices in Table 5.2 in Part I of this paper, and also ill-conditioned matrices up to dimension 1000.

In this test, (2.11) was almost always the best bound and (2.10) was a little weaker. The bound (2.8) was usually worse than (2.11) by a factor between 1.5 and 2, in a few cases up to a factor 15. Hence the results are as expected.

The normwise bound (2.3) often failed for ill-conditioned matrices and $t = e$ because of $\|F\|_\infty \ge 1$. When choosing the Perron vector for t, the bound (2.3) was usually worse than (2.11) by a factor around 2, sometimes up to a factor 20. The mentioned factors occur only for very ill-conditioned matrices, not too far from $\mathbf{u}^{-1}$. For matrices with moderate condition number all bounds are almost identical.

Besides (2.9) we may use $C^{-1} = I + FC^{-1}$ to deduce that $|\Delta| = |C^{-1}c| \le \epsilon$ implies $|\Delta| \le |c| + |F|\epsilon$, which means $|\Delta| \le \min\big(\epsilon, |c| + |F|\epsilon\big)$. Applying this a few times to the discussed bounds, all final bounds were not too far apart. Since this improvement costs only some $\mathcal{O}(n^2)$ operations, a good choice seems to be to use (2.8), which computes in $\mathcal{O}(n)$ operations, and to improve it as mentioned.

Theorem 2.1. *Let $A, R \in \mathbb{R}^{n\times n}$ and $b, \widetilde{x} \in \mathbb{R}^n$ be given. Let $0 < v \in \mathbb{R}^n$ be such that $u := \langle RA\rangle v > 0$. Denote by $\langle RA\rangle := D - E$ the splitting of $\langle RA\rangle$ into diagonal and off-diagonal parts, and define $w \in \mathbb{R}^n$ by (2.6). Then A and R are non-singular, and*

$$|A^{-1}b - \widetilde{x}| \le (D^{-1} + vw^T)|c| \quad \text{for } c := R(b - A\widetilde{x}). \tag{2.12}$$

Moreover, $|A^{-1}b - \widetilde{x}| \le \epsilon$ implies

$$|A^{-1}b - \widetilde{x}| \le D^{-1}\big(|c| + E\epsilon\big) \quad \text{and} \quad |A^{-1}b - \widetilde{x}| \le |c| + |I - RA|\epsilon. \tag{2.13}$$

2.1. Verification of H-property

It remains to find a positive vector v with $u := \langle C\rangle v > 0$, which means to prove that $C = RA$ is an H-matrix. There is a vast amount of literature on this problem, see, for example, [8] and the references therein. The optimal choice of v can be seen as follows.

Since RA is not too far from the identity matrix, the vector $e := (1, \ldots, 1)^T \in \mathbb{R}^n$ seems a proper choice for v and is often used, for example in [6]. However, this means that $\langle C\rangle = D - E$ is strictly diagonally dominant, which need not be the case due to corruption of R by rounding errors. A better choice is $D^{-1}e$, but even then for ill-conditioned matrices RA it may happen that RA is an H-matrix, whereas this fact cannot be verified by both choices of the vector v. This is not uncommon, see Fig. 3.1 in Section 3.2. After spending considerable effort to compute RA, it is particularly annoying to fail to prove RA to be an H-matrix only because of some standard choice of v.

Therefore it is worth to spend a few vector iterations to adapt the vector v to the particular circumstances. The matrix $\langle C\rangle = D - E$ is an M-matrix if and only if $r := \varrho(D^{-1}E) < 1$. But $D^{-1}E$ is a nonnegative matrix, thus its spectral radius

corresponds to an eigenvalue, the Perron root. Usually $D^{-1}E$ is at least nonnegative irreducible if not positive, so that the corresponding eigenvector is positive, and a power iteration

$$v^{(k+1)} := D^{-1}Ev^{(k)} \quad \text{for } v^{(0)} := D^{-1}e \tag{2.14}$$

implies that $\max_i v_i^{(k+1)}/v_i^{(k)}$ decreases monotonically to the Perron root [9]. Assume $D^{-1}Ev^{(k)} \approx rv^{(k)}$ for $r < 1$, then

$$\langle C\rangle v^{(k)} = D(I - D^{-1}E)v^{(k)} \approx (1-r)Dv^{(k)} > 0. \tag{2.15}$$

Therefore, starting with $v := D^{-1}e$, we perform a few power iterations to find a positive vector v satisfying $\langle C\rangle v > 0$. The computational cost for one iteration is one matrix-vector multiplication plus $\mathcal{O}(n)$ operations. The resulting vector v is required to be positive, which is not true if a row of E is identically zero. This is taken care of by adding some small positive constant ε to the iterates. The Matlab code needs directed rounding, therefore it is deferred to Section 3, see Algorithm 3.2 (`MVector`).

The vector u determines the size of w in (2.6), and hence the quality of the upper bounds of $\langle C\rangle^{-1}$ used in (2.11) and also in (2.3). Therefore we add in practice another one or two iterations if the vector u is too small in magnitude. It adds few operations, but may improve the bounds significantly. To the optimality of the choice of v consider the following lemma.

Lemma 2.2. *Let a matrix $C \in \mathbb{C}^{n\times n}$ be given, and denote by $N \in \mathbb{R}^{n\times n}$ an arbitrary irreducible non-negative matrix. If $C_{\nu\nu} = 0$ for some $1 \le \nu \le n$, then C is not an H-matrix. Suppose $C_{\nu\nu} \ne 0$ for all $1 \le \nu \le n$. Then for small enough positive ε the following is true.*

For the splitting $D - E := \langle C\rangle - \epsilon N$ into diagonal and off-diagonal parts, the matrix $M := D^{-1}E$ has a (unique) positive eigenvector v to its spectral radius, and C is an H-matrix if and only if $\langle C\rangle v$ is positive. If the off-diagonal part of $\langle C\rangle$ is irreducible, then the above is true for $\varepsilon = 0$. In that case, for any positive vector w, $\langle C\rangle w > 0$ implies

$$\min_{1\le i\le n} \frac{[\langle C\rangle w]_i}{w_i} < \min_{1\le i\le n} \frac{[\langle C\rangle v]_i}{v_i}. \tag{2.16}$$

Proof. If C is an H-matrix, then $\langle C\rangle w > 0$ for some positive w, and therefore $C_{\nu\nu} \ne 0$ for all $1 \le \nu \le n$. For small enough positive ε, the matrix C is an H-matrix if and only $\langle C\rangle - \epsilon N$ is an M-matrix. Assume such an ε be given and $C_{\nu\nu} \ne 0$ for all $1 \le \nu \le n$, so that D is invertible.

The off-diagonal part of $\langle C\rangle$ is non-positive by definition, so that E is non-negative irreducible, and so is $M = D^{-1}E$. By Perron–Frobenius theory [9] there is a unique positive eigenvector v of M corresponding to its spectral radius. If $\langle C\rangle v > 0$, then C is an H-matrix. If conversely C is an H-matrix, then $D - E = \langle C\rangle - \epsilon N$ is an M-matrix and $\lambda := \varrho(D^{-1}E) < 1$, so that

$$\langle C\rangle v = D(I - D^{-1}E)v + \varepsilon Nv = D(1-\lambda)v + \varepsilon Nv > 0. \tag{2.17}$$

If E is irreducible, then the assertions are valid for $\varepsilon = 0$. In that case Perron–Frobenius theory implies

$$\min_{1\le i\le n} \frac{\left[D^{-1}Ew\right]_i}{w_i} < \lambda < \max_{1\le i\le n} \frac{\left[D^{-1}Ew\right]_i}{w_i}$$

for any positive vector $w \ne v$. Hence there is an index k with $\left[D^{-1}Ew\right]_k > \lambda w_k$. Using $\langle C\rangle = D(I - D^{-1}E)$ and (2.17) with $\varepsilon = 0$ we conclude

$$\frac{[\langle C\rangle w]_k}{w_k} < \frac{[D(1-\lambda)w]_k}{w_k} = (1-\lambda)D_{kk} = \frac{[\langle C\rangle v]_k}{v_k} = \frac{[\langle C\rangle v]_i}{v_i}$$

with the last equality being valid for all $i \in \{1, \ldots, n\}$. The lemma is proved. □

In practice, a choice for N is the matrix of all 1's. The ϵ-perturbation may be avoided by using the usual splitting $D - E := \langle C\rangle$ and applying the theorem to the irreducible normal form of $D^{-1}E$. For our purposes, however, Algorithm 3.2 (`MVector`) is sufficient.

3. Rigorous error bounds using directed rounding

In Part I of this paper we discussed the computation of rigorous bounds using solely floating-point arithmetic in rounding to nearest. Special care was necessary to cover all rounding errors, in particular for results in the underflow range.

This is simpler when using directed rounding. Denote by $\mathbb{F}$ a set of p-bit binary floating-point numbers including infinity and NaN, i.e.

$$\mathbb{F} = \{\, M \cdot 2^{e-p+1} \mid M, e \in \mathbb{Z}, |M| \le 2^p - 1, e_{\min} \le e \le e_{\max}\} \cup \{-\infty, +\infty, \text{NaN}\}. \tag{3.1}$$

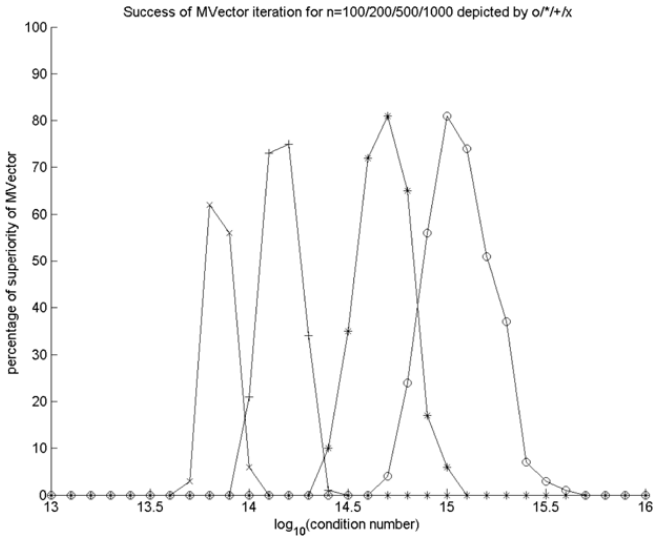

Fig. 3.1. Percentage that $\langle \mathtt{R} \cdot \mathtt{A} \rangle x > 0$ is satisfied for choosing the Perron vector computed by Algorithm 3.2 (`Mvector`) for x, but not satisfied for $x := (1, \ldots, 1)^T$. Graphs for random matrices of dimensions $n = 100$ (o), $n = 200$ ($*$), $n = 500$ ($+$) and $n = 1000$ ($\times$) and condition numbers ranging from 10^{13} to 10^{16} are shown.

For details see [10] and Part I of this paper. We consider three different ways to round real numbers into floating-point numbers: rounding to nearest $\mathrm{fl}_\square$, rounding to downwards $\mathrm{fl}_\triangledown$ and rounding to upwards $\mathrm{fl}_\triangle$. For $x \in \mathbb{R}$ they are defined by

$$\begin{aligned} &|\mathrm{fl}_\square(x) - x| = \min\{|f - x| : f \in \mathbb{F}\}, \\ &\mathrm{fl}_\triangledown(x) := \max\{f \le x : f \in \mathbb{F}\}, \\ &\mathrm{fl}_\triangle(x) := \min\{f \ge x : f \in \mathbb{F}\}, \end{aligned} \tag{3.2}$$

where the ambiguity in the definition of $\mathrm{fl}_\square$ is resolved by rounding ties to even. For $a, b \in \mathbb{F}$ and $\circ \in \{+, -, \cdot, /\}$ these rounding functions define floating-point operations $\mathrm{fl}_\square(a \circ b)$, $\mathrm{fl}_\triangledown(a \circ b)$ and $\mathrm{fl}_\triangle(a \circ b)$, respectively, mapping $\mathbb{F} \times \mathbb{F}$ into $\mathbb{F}$. All those operations are mandatory in the IEEE 754 floating-point standard [11,12], and most computers comply with this standard. For $a, b \in \mathbb{F}$ it follows

$$\mathrm{fl}_\triangledown(a \circ b) \le a \circ b \le \mathrm{fl}_\triangle(a \circ b) \quad \text{and} \quad \mathrm{fl}_\triangledown(a \circ b) = \mathrm{fl}_\triangle(a \circ b) \Leftrightarrow a \circ b \in \mathbb{F}. \tag{3.3}$$

The default in Matlab is rounding to nearest. Fortunately, directed rounding is accessible in Matlab through INTLAB [1], the Matlab toolbox for reliable computing. As has been mentioned, the toolbox INTLAB is entirely written in Matlab and thus portable in many environments. The command `setround(i)` changes the rounding mode according to the following table:

Matlab command	effect
`setround(0)`	rounding to nearest
`setround(−1)`	rounding to downwards
`setround(1)`	rounding to upwards

(3.4)

The `setround` command changes the control word of the processor so that henceforth *all* floating-point operations are executed in the specified rounding mode until the next `setround` command. This implies that for given floating-point numbers `a,b,c` the code

```
setround(−1)
pinf = a ∗ b − c;
setround(1)
psup = a ∗ b − c;
```

(3.5)

computes floating-point numbers `pinf`, `psup` satisfying

$$\mathtt{pinf} \le \mathtt{a} \cdot \mathtt{b} - \mathtt{c} \le \mathtt{psup}, \tag{3.6}$$

where $\mathtt{a} \cdot \mathtt{b} - \mathtt{c} \in \mathbb{R}$ denotes the true real result. Note that (3.5) always implies (3.6), also in the presence of overflow or underflow (if overflow occurs the bounds are, however, infinite or NaN). With a little thinking it becomes clear that the same code (3.5) is also working for vectors and matrices of compatible size, in which case `pinf` and `psup` are quantities of the corresponding dimension satisfying (3.6). Note that for $c - a \cdot b$ this approach would not work correctly.

Directed rounding is accessible in standard Matlab on many architectures using the undocumented command `feature ('setround',rnd)` with values 0.5, $-\infty$ and ∞ of `rnd` for switching the rounding to nearest, downwards and upwards, respectively.

3.1. Executable code implementing Theorem 2.1 using directed rounding

The following Algorithm 3.1 computes an approximate solution xs together with an error bound err such that $|\mathtt{A}^{-1}\mathtt{b} - \mathtt{xs}| \leq \mathtt{err}$, and proves non-singularity of the input matrix A. It is based on Theorem 2.1. The exposition would become easier and more readable when using the interval operations as provided by INTLAB [1]. We avoid this dependency for the moment and present the algorithm using only directed rounding, i.e. pure Matlab code. From the next Section 4 on, we use interval operations because the algorithms become more involved and would become difficult to read when using only directed rounding.

Algorithm 3.1. Approximate solution xs of $Ax = b$ with error bound err such that $|\mathtt{A}^{-1}\mathtt{b} - \mathtt{xs}| \leq \mathtt{err}$ is rigorously satisfied (together with the proof of non-singularity of A).

```
 1 function [xs,err] = LssErrBndDirRdg(A,b)
 2   setround(0)                                % rounding to nearest
 3   err = repmat(NaN,size(b)); n = size(A,2); % initialization
 4   R = inv(A);                                % approximate inverse of A
 5   xs = ResidIter(A,b,R);                     % improved approximate solution
 6   [rinf,rsup] = Dot2DirRdg([A b],[xs;-1],1);   % inclusion of A*xs-b
 7   setround(+1)                               % rounding to upwards
 8   rmid = rinf + 0.5*(rsup-rinf);             % rmid >= (rinf+rsup)/2
 9   delta = abs(R)*(rmid-rinf);                % delta >= |R|r with r:=(rsup-rinf)/2
10   csup = R*rmid + delta;                     % upper bound of R(A*xs-b)
11   Csup = R*A;                                % upper bound of RA
12   setround(-1)                               % rounding to downwards
13   cinf = R*rmid - delta;                     % [cinf,csup] inclusion of R(A*xs-b)
14   Cinf = R*A;                                % [Cinf,Csup] inclusion of RA
15   c = max(abs(cinf),abs(csup));              % upper bound of |R(A*xs-b)|
16   d = diag(Cinf);                            % lower bound of diag(RA)
17   if any(d<=0), return, end                  % C not H-matrix, no inclusion
18   E = max(abs(Cinf),abs(Csup)); E(1:n+1:n^2) = 0;  % <C>=D-E  with D,E>=0, D:=diag(d)
19   [u,v] = MVector(E,d);                      % vector iteration to verify M-property
20   if any(u<=0) || any(v<=0), return, end     % verification of M-property failed
21   setround(1)                                % rounding to upwards
22   w = max(E./(u*d'));                        % upper bound u*w >= E*D^-1 (w row)
23   dinv = 1./d;                               % upper bound of diag(D^-1)
24   err = dinv.*c + v*(w*c);                   % error bound for xs
25   N = inf;                                   % initialize constant
26   for iter=1:15                              % at most 15 improvements of error bound
27     err = min(err, dinv.*(c + E*err));       % improved error bound
28     Nold = N; N = max(err);                  % norm of correction
29     if N >= 0.99*Nold, break, end            % stop iteration if no improvement
30   end
31   setround(0)                                % rounding to nearest
```

3.2. Proof of correctness of Algorithm 3.1 (*LssErrBndDirRdg*)

Let an $n \times n$-matrix A and right hand side $\mathtt{b} \in \mathbb{F}^n$ with floating-point entries be given. Algorithm 3.1 is executable Matlab/INTLAB code, so all operations are floating-point operations with a floating-point result. In contrast, *all of the operations in the following analysis are the exact real operations*. For example, using R as computed in line 4, $C := \mathtt{R} \cdot \mathtt{A} \in \mathbb{R}^{n \times n}$ denotes the true matrix product of the (floating-point) matrices R and A. In general, $C \notin \mathbb{F}^{n \times n}$.

After making sure that the rounding is set to nearest in line 2, an approximate inverse R of A and an approximate solution xs is computed in lines 4 and 5, the latter improved by a residual iteration using Algorithm 4.13 in Part I of this paper. Nothing is required on the quality of R and xs for the following theoretical analysis. Assume for the moment that the quantities $\mathtt{rinf}, \mathtt{rsup} \in \mathbb{F}^n$ computed in line 6 satisfy

$$\mathtt{rinf} \leq \mathtt{A} \cdot \mathtt{xs} - \mathtt{b} \leq \mathtt{rsup}. \tag{3.7}$$

We come to that at the end of this section. We split the computation of delta in line 9 into

```
rrad = rmid-rinf; delta = abs(R)*rrad;
```

both computed in rounding to upwards as specified in line 7. Then using line 8

$$(\mathtt{rinf} + \mathtt{rsup})/2 = \mathtt{rinf} + (\mathtt{rsup} - \mathtt{rinf})/2 \le \mathtt{rmid} \quad \text{and} \quad \mathtt{rmid} - \mathtt{rinf} \le \mathtt{rrad},$$

so that

$$\mathtt{rmid} - \mathtt{rrad} \le \mathtt{rinf} \quad \text{and} \quad \mathtt{rsup} \le 2 \cdot \mathtt{rmid} - \mathtt{rinf} \le \mathtt{rmid} + \mathtt{rrad}. \tag{3.8}$$

Define $\mu := \mathtt{rmid}$ and $\rho := \mathtt{rrad}$, then (3.7) implies $\mathtt{A} \cdot \mathtt{xs} - \mathtt{b} = \mu + r$ for $|r| \le \rho$. We use the idea in (3.5) and (3.6) and rounding to upwards to conclude

$$\begin{aligned} &\mathtt{R} \cdot r \le |\mathtt{R} \cdot r| \le |\mathtt{R}| \cdot \mathtt{rrad} \le \mathtt{delta} \\ &\mathtt{R} \cdot (\mathtt{A} \cdot \mathtt{xs} - \mathtt{b}) = \mathtt{R} \cdot (\mu + r) \le \mathtt{R} \cdot \mathtt{rmid} + \mathtt{delta} \le \mathtt{csup} \\ &\mathtt{R} \cdot \mathtt{A} \le \mathtt{Csup}. \end{aligned} \tag{3.9}$$

After switching the rounding to downwards in line 12 we obtain from lines 13 and 14

$$\begin{aligned} &\mathtt{cinf} \le \mathtt{R} \cdot \mathtt{rmid} - \mathtt{delta} \le \mathtt{R} \cdot (\mu + r) = \mathtt{R} \cdot (\mathtt{A} \cdot \mathtt{xs} - \mathtt{b}) \\ &\mathtt{Cinf} \le \mathtt{R} \cdot \mathtt{A}. \end{aligned} \tag{3.10}$$

This nice implicit transformation of infimum–supremum into midpoint-radius is due to Oishi and Rump [13]. The operations in lines 15 and 16 cause no rounding error, so that after line 17

$$|\mathtt{R} \cdot (\mathtt{A} \cdot \mathtt{xs} - \mathtt{b})| \le \mathtt{c} \text{ and } \mathrm{diag}(\mathtt{R} \cdot \mathtt{A}) \ge \mathtt{d} > 0 \tag{3.11}$$

is satisfied. After extracting the diagonal of E in line 18 we have

$$\langle \mathtt{R} \cdot \mathtt{A} \rangle \ge \mathrm{diag}(\mathtt{d}) - \mathtt{E} \quad \text{with } \mathtt{d} > 0 \quad \text{and} \quad \mathtt{E} \ge 0. \tag{3.12}$$

For the true splitting $\langle \mathtt{R} \cdot \mathtt{A} \rangle = D - E$ this implies

$$D \ge \mathrm{diag}(\mathtt{d}) \quad \text{and} \quad E \le \mathtt{E}, \tag{3.13}$$

so that the error bounds in Theorem 2.1 remain valid when replacing D and E by the computed quantities diag(d) and E, respectively.

The computation of the vectors u and v in line 19 is based on the described Perron iteration for $D^{-1}E$ using the computed quantities d and E. The corresponding Algorithm 3.2 (MVector) is given after the proof. It computes vectors u and v satisfying $(\mathrm{diag}(\mathtt{d}) - \mathtt{E}) \cdot \mathtt{v} \ge \mathtt{u}$, so that $\langle \mathtt{R} \cdot \mathtt{A} \rangle = D - E$ implies

$$\langle \mathtt{R} \cdot \mathtt{A} \rangle \cdot \mathtt{v} \ge (\mathrm{diag}(\mathtt{d}) - \mathtt{E}) \cdot \mathtt{v} \ge \mathtt{u}. \tag{3.14}$$

Thus arriving in line 21 means $\mathtt{u} > 0$ and $\mathtt{v} > 0$, so that $\mathtt{R} \cdot \mathtt{A}$ is an H-matrix, and A (and R) are proved to be non-singular. Using (3.13) we see that the (row) vector w computed in rounding to upwards in line 22 satisfies

$$\mathtt{w}_k \ge \max_i \frac{\mathtt{E}_{ik}}{\mathtt{u}_i \mathtt{d}_k} \ge \max_i \frac{\left(ED^{-1}\right)_{ik}}{\mathtt{u}_i} \quad \text{for all } 1 \le k \le n, \tag{3.15}$$

so that

$$\mathtt{u}_i \mathtt{w}_k \ge \left(ED^{-1}\right)_{ik} \quad \text{for all } 1 \le i, k \le n. \tag{3.16}$$

It follows that err computed in line 24 in rounding to upwards is an upper bound of the right hand side of (2.12), so that indeed $|\mathtt{A}^{-1}\mathtt{b} - \mathtt{xs}| \le \mathtt{err}$. The correctness of the final bound follows by (2.13) and the rounding to upwards.

The Perron iteration based on Section 2.1 is performed by the following algorithm MVector.

Algorithm 3.2. Perron vector iteration to find the optimal vector $v > 0$ proving $\langle C \rangle = D - E$ to be an M-matrix by $\langle C \rangle v > 0$. To save memory, D is represented by its diagonal.

```
function [u,v] = MVector(E,d)
% Vector iteration on M=D-E, D,E>=0, D:=diag(d), to verify M-property
  setround(-1)                              % Vector iteration in rounding downwards
  vnew = 1./d;                              % first guess
  minu = -inf;                              % initialization
  for iter=1:15                             % at most 15 iterations
    v = vnew;  minuold = minu;              % update
    w = E*(-v);
    u = d.*v + w;                           % vector iteration, u <= M*v
    minu = min(u);                          % new minimum of M*v
    vnew = -w./d + eps;                     % update of guess of H-vector
    r = vnew./v;                            % ratio new and old Perron vector
    if ( max(r)<1.001*min(r) ) && ( ( minu>0 ) || ( minu<minuold ) )
      return                                % sufficiently accurate
    end
  end
```

If successful, the output are two positive vectors u and v with $(\mathtt{D}-\mathtt{E})\cdot\mathtt{v}\geq\mathtt{u}>0$, so that $\mathtt{D}-\mathtt{E}$ is an H-matrix provided $\mathtt{E}\geq 0$. This is true because the entire iteration is performed in rounding to downwards, so that always $\mathtt{w}\leq\mathtt{E}\cdot(-\mathtt{v})=-\mathtt{E}\cdot\mathtt{v}$ and

$$\mathtt{u}\leq\mathtt{D}\cdot\mathtt{v}+\mathtt{w}\leq(\mathtt{D}-\mathtt{E})\cdot\mathtt{v}\leq\langle\mathtt{R}\cdot\mathtt{A}\rangle\mathtt{v}.$$

As has been mentioned, it may happen for ill-conditioned matrices that $\langle\mathtt{R}\cdot\mathtt{A}\rangle x>0$ is satisfied for choosing an approximation of the Perron vector of $D^{-1}E$ for x, but is not satisfied for the standard choice $x:=(1,\ldots,1)^T$. The percentage of such cases for random matrices of different dimensions and condition numbers is displayed in Fig. 3.1.

It remains to specify Algorithm Dot2DirRdg and to prove the correctness of (3.7). Using directed rounding it is easy to rewrite Algorithm 3.4 (Dot2Near) from Part I:

Algorithm 3.3. Approximation of the matrix product $\mathtt{A}\cdot\mathtt{B}$ for $\mathtt{A}\in\mathbb{F}^{m\times k}$, $\mathtt{B}\in\mathbb{F}^{k\times n}$ "as if" accumulated in twice the working precision and rounded to working precision with rigorous error term.

```
 1   function [res,ressup] = Dot2DirRdg(A,B,Incl)
 2     setround(0)                              % rounding to nearest
 3     [p,e] = TwoProduct(A(:,1),B(1,:));       % error-free transformation of first product
 4     if Incl, einf = e; esup = e; end         %      (underflow is taken care of later)
 5     k = size(A,2);                           % inner dimension
 6     for i=2:k                                % matrix product by rank-1 updates
 7       [h,r] = TwoProduct(A(:,i),B(i,:));     % error-free transformation of i-th product
 8       [p,q] = TwoSum(p,h);                   % error-free transformation of accumulated sum
 9       if Incl                                % rigorous error bounds to be computed
10         setround(-1); einf = einf+(q+r);     % lower bound of sum of errors
11         setround(+1); esup = esup+(q+r);     % upper bound of sum of errors
12         setround(0)                          % rounding to nearest
13       else                                   % approximation to be computed
14         e = e + (q + r);                     % approximate accumulation of errors
15       end
16     end
17     if Incl                                  % output inclusion [res,ressup]
18       setround(+1)                           % rounding to upwards
19       delta = max(1.5*k*eps,1)*realmin;      % cover possible underflow
20       ressup = p + (esup + delta);           % upper bound of A*B
21       setround(-1);                          % rounding to downwards
22       res = p + (einf - delta);              % lower bound of A*B
23       setround(0)                            % rounding to nearest
24     else                                     % output approximation res
25       res = p + e;                           % extra-precise approximation
26     end
```

We claim that the call $\mathtt{res}=\mathtt{Dot2DirRdg}(\mathtt{A},\mathtt{B},0)$ computes an approximation res of the product $\mathtt{A}\cdot\mathtt{B}$ with extra-precise accumulation of the dot products, and after the call $[\mathtt{res},\mathtt{err}]=\mathtt{Dot2DirRdg}(\mathtt{A},\mathtt{B},1)$ the computed quantities res and ressup satisfy

$$\mathtt{res}\leq\mathtt{A}\cdot\mathtt{B}\leq\mathtt{ressup}. \tag{3.17}$$

For $\mathtt{Incl}=0$, the approximation res is computed as in Algorithm 3.4 (Dot2Near). For the computation of the rigorous error bounds, assume for the moment that no underflow occurs. Then both TwoProduct and TwoSum in lines 3, 7 and 8 are error-free transformations, see (3.1) in Part I of this paper. If in line 14 there would be no rounding error, then after the line ending the loop in line 16 the true sum $\mathtt{p}+\mathtt{e}$ would be equal to the true product $\mathtt{A}\cdot\mathtt{B}$. But in lines 10, 11, 20 and 22 the summations are performed with directed rounding, and the result (3.17) follows. Possible underflow is covered in line 19 according to (3.1) in Part I of this paper, so that (3.17) is always true. Note that setting the rounding to nearest in line 12 is necessary to ensure that TwoProduct and TwoSum are indeed error-free transformations.

We proved the following result.

Theorem 3.4. *Let* xs *and* err *be the results of Algorithm 3.1 (*LssErrBndDirRdg*) applied to* $\mathtt{A}\in\mathbb{F}^{n\times n}$ *and* $\mathtt{b}\in\mathbb{F}^n$. *If Algorithm 3.1 ends successfully, i.e.* err *is not a vector of* NaN*'s, then the input matrix* A *is non-singular and*

$$|\mathtt{A}^{-1}\mathtt{b}-\mathtt{xs}|\leq\mathtt{err}. \tag{3.18}$$

3.3. Improvement of Algorithm 3.1

In this subsection we discuss an algorithmic improvement of Algorithm 3.1 which – for simplicity – was not mentioned before. Theorem 2.1, which is implemented by Algorithm 3.1, is based on the splitting $\langle RA\rangle=D-E$ so that error bounds

for the matrix product RA are necessary. The standard error estimate [14] for a dot product x^Ty for $x, y \in \mathbb{R}^k$ evaluated in floating-point is

$$\left|\mathrm{fl}(x^Ty) - x^Ty\right| \le \gamma_k |x^T \parallel y| + k\mathrm{eta}/2 \tag{3.19}$$

using $\gamma_k := k\mathbf{u}/(1-k\mathbf{u})$, where the extra term $k$eta/2 covers underflow; in IEEE 754 double precision (binary64) we have $\mathbf{u} = 2^{-53}$ and eta $= 2^{-1074}$. For $R, A \in \mathbb{F}^{n\times n}$ it follows immediately

$$|\mathrm{fl}(RA) - RA| \le \gamma_n |R \parallel A| + n\mathrm{eta}/2 \cdot ee^T. \tag{3.20}$$

However, two matrix multiplications are necessary to bound RA, the same as for calculating RA in rounding downwards and upwards. But the application of Theorem 2.1 requires only a lower bound of the diagonal of $|RA|$ and an upper bound of the off-diagonal elements of $|RA|$. More precisely, the off-diagonal elements in E are only needed for the Perron iteration in Algorithm 3.2 (`MVector`) and the rigorous computation of w satisfying (2.6). Both can be covered by the following code requiring only one matrix multiplication:

```
1   setround(-1);                                % rounding to downwards
2   Cinf = R*A;                                  % lower bound Cinf <= RA
3   d = diag(Cinf);                              % lower bound of diag(RA)
4   if all(d>0)                                  % C may be H-matrix
5     setround(1);                               % rounding to upwards
6     e = ones(n,1);                             % vector (1,...,1)^T
7     g = n*eps/(-(n*eps-1));                    % upper bound for gamma_2n
8     Ee = ( abs(Cinf)*e-d ) + ( g*(abs(R)*(abs(A)*e)-d) + max(n^2*eps,1)*realmin*e );
```

The rounding mode in line 1 implies `Cinf` $\le RA$, so that $0 <$ `d` $\le \mathrm{diag}(RA)$ in line 5. Rounding to downwards defines floating-point operations with relative rounding error unit $2\mathbf{u}$ and maximum error eta in the underflow range rather than eta/2. Therefore (3.20) and $\langle RA\rangle = D - E$ imply

$$E = |E| \le |\mathtt{Cinf}| - D + \frac{2n\mathbf{u}}{1-2n\mathbf{u}}\left(|R \parallel A| - D\right) + n\mathrm{eta}\cdot ee^T. \tag{3.21}$$

Thus observing the definition eps $:= 2^{-52} = 2\mathbf{u}$ in Matlab, `realmin` $= \frac{1}{2}\mathbf{u}^{-1}$eta $=$ `eps`$^{-1}$eta, the rounding mode and a little thinking proves

$$0 < \mathtt{d} \le D = \mathrm{diag}(RA) \quad \text{and} \quad Ee \le \mathtt{Ee}. \tag{3.22}$$

Up to now only one matrix multiplication `Cinf` $=$ `R` $*$ `A` was necessary by avoiding to calculate $|R \parallel A|$ explicitly. This is also possible in the further calculations. The only occurrence of E in Algorithm 3.2 (`MVector`) is `w` $=$ `E` $*$ $(-$`v`$)$ in line 8. Replacing this line by `w` $=$ $(-$`max(v)`$)$ $*$ `Ee`, setting $\mu := \max_i(v_i)$, observing $E, v \ge 0$ and the rounding to downwards ensures `w` $\le -\mu \cdot$ `Ee` $\le -E\cdot(\mu e) \le -E$v as desired. Furthermore E is only used in lines 22 and 27 of Algorithm 3.1 (`LssErrBndDirRdg`). Replace line 22 by

```
uinv = 1./u;                                % upper bound of u_i^-1
w = max(uinv(:,ones(1,n)).*abs(Cinf)).*dinv' - uinv' ;
w = w + (g*(((uinv'*abs(R))*abs(A)).*dinv') + max(n*eps*max(uinv)*max(dinv),1)*realmin);
```

Here `uinv(:,ones(1,n))` is the $n \times n$-matrix with columns `uinv`, so that `uinv(:, ones(1, n))`. $*$ `abs(Cinf)` is an upper bound of $\tilde{D}|$`Cinf`$|$ for $\tilde{D}$ denoting the diagonal matrix with entries $\mathtt{u}_\nu^{-1}$. The maximum in the first computation of `w` is the (row) vector of the maxima of columns, so that by $\mathtt{u}_\nu^{-1} \le \mathtt{uinv}_\nu$ and $D_{\nu\nu}^{-1} \le \mathtt{dinv}_\nu$ the first computed `w` satisfies

$$\mathtt{w}_k \ge \max_i \frac{|\mathtt{Cinf}|_{ik}}{\mathtt{u}_i D_{kk}} - \mathtt{u}_k^{-1} = \max_i \frac{(|\mathtt{Cinf}| - D)_{ik}}{\mathtt{u}_i D_{kk}} = \max_i \frac{\left((|\mathtt{Cinf}| - D)D^{-1}\right)_{ik}}{\mathtt{u}_i}$$

for $1 \le k \le n$. For the error term we have

$$\gamma_{2n}\cdot \max_i \frac{(|R \parallel A| - D)_{ik}}{\mathtt{u}_i D_{kk}} + \max_i \frac{n\mathrm{eta}}{\mathtt{u}_i D_{kk}} \le \left[\mathtt{uinv}^T |R \parallel A| D^{-1}\right]_k + n\left(\max_\nu \mathtt{uinv}_\nu\right)\left(\max_\nu \mathtt{dinv}_\nu\right)\cdot \mathrm{eta}.$$

Hence, putting things together, (3.21) implies

$$\mathtt{w}_k \ge \max_i \frac{(ED^{-1})_{ik}}{\mathtt{u}_i} \quad \text{for } 1 \le k \le n,$$

so that $(ED^{-1})_{ik} \le \mathtt{u}_i \mathtt{w}_k$ is satisfied as required in the analysis following (2.6). Finally, replacing `E` $*$ `err` in line 27 by `Ee` $*$ `max(err)`, an upper bound of the estimate in (2.13) is computed. This shows that with only one matrix multiplication `Cinf` $=$ `R` $*$ `A` verified bounds can be computed.

The quality of the bounds is almost identical to those by the original Algorithm 3.1 (LssErrBndDirRdg) provided that Algorithm 3.2 (MVector) succeeds to compute vectors $\mathtt{u}, \mathtt{v} \in \mathbb{F}^n$ with $\langle RA\rangle\mathtt{v} \geq \mathtt{u} > 0$. This may fail due to the relaxed estimate of the upper bound of E in (3.21). In that case $\mathtt{Csup} = \mathtt{R} * \mathtt{A}$ is computed as in Algorithm 3.1, a new Perron vector is computed by MVector based on the improved E, and we continue as before.

The advantage is that for not too ill-conditioned problems one matrix multiplication can be saved. In any case, whether the approach succeeds or not, only a few additional $\mathcal{O}(n^2)$ operations are necessary. Asymptotically, if successful, the computing time decreases by about one third. In practice, as has been mentioned, the performance of matrix multiplications is better than that of matrix inversion, and also some interpretation overhead is added. For timing results see Fig. 8.2.

4. Linear systems with data afflicted with tolerances

Denote by $\mathbb{IR}$ the set of non-empty closed and bounded intervals

$$\mathbf{x} = [x_1, x_2] = \{x \in \mathbb{R} : x_1 \leq x \leq x_2\}. \tag{4.1}$$

Operations $\circ : \mathbb{IR} \times \mathbb{IR} \to \mathbb{IR}$ for $\circ \in \{+, -, \cdot, /\}$ are defined by

$$\mathbf{x}, \mathbf{y} \in \mathbb{IR} : \mathbf{x} \circ \mathbf{y} := \bigcap\{\mathbf{z} \in \mathbb{IR} : Z \subseteq \mathbf{z}\} \quad \text{for } Z := \{x \circ y : x \in \mathbf{x}, y \in \mathbf{y}\} \tag{4.2}$$

provided $0 \notin \mathbf{y}$ in case of division. For these scalar operations always $\mathbf{x} \circ \mathbf{y} = Z$, i.e. there is no difference to the power set operation, and obviously the inclusion property

$$x \in \mathbf{x}, y \in \mathbf{y} \Rightarrow x \circ y \in \mathbf{x} \circ \mathbf{y} \tag{4.3}$$

holds true. Intervals with equal endpoints define the natural embedding of $\mathbb{R}$ into $\mathbb{IR}$. Using the partial ordering we define the sets $\mathbb{IR}^n$ and $\mathbb{IR}^{m\times n}$ of non-empty and compact interval vectors and matrices similar to (4.1), respectively, with the natural embedding as before. The same definition (4.2) is used for operations between compatible vector and matrix quantities. The sets $\mathbb{IF}$, $\mathbb{IF}^n$ and $\mathbb{IF}^{m\times n}$ are defined in the same way except that the bounds are floating-point quantities, for example

$$\mathbf{A} = [A_1, A_2] = \{A \in \mathbb{R}^{m\times n} : A_1 \leq A \leq A_2\} \in \mathbb{IF}^{m\times n} \quad \text{for } A_1, A_2 \in \mathbb{F}^{m\times n}. \tag{4.4}$$

Note that although the bounds are floating-point quantities, the intervals comprise of all real quantities within the bounds. Obviously, $\mathbb{IF}^{m\times n} \subseteq \mathbb{IR}^{m\times n}$. Again, operations on $\mathbb{IF}$, $\mathbb{IF}^n$ and $\mathbb{IF}^{m\times n}$ can be defined as in (4.2), and again the important inclusion property (4.3) is true. For more details see [15].

In view of practical applications it is not necessary and usually not wise to insist on narrowest inclusions as in (4.2). In the vast majority of applications (as in this paper) only the inclusion property (4.3) is used. This allows the design of fast interval operations based on BLAS libraries (cf. [16]). Based on that, all interval operations are available in INTLAB [1]. As an example, for matrices $\mathtt{A}, \mathtt{B} \in \mathbb{F}^{n\times n}$, an inclusion [Cinf, Csup] of $\mathtt{A} \cdot \mathtt{B}$ is computed by

```
setround(-1); Cinf = A*B;
setround(+1); Csup = A*B;
```

This inclusion is not the best possible, but satisfies the inclusion property (4.3). This approach was used in lines 14 and 11 of Algorithm 3.1 (LssErrBndDirRdg). Real quantities may be replaced by an including interval with floating-point endpoints, so that interval operations allow rigorous error bounds for operations with real numbers.

INTLAB makes use of the operator concept in Matlab by defining a data type intval. For a floating-point (scalar, vector or matrix) quantity x, the type cast intval(x) is a variable of type intval with left and right bound x. Moreover, infsup(xinf,xsup) checks xinf $\leq$ xsup and defines an interval with the left and right bounds xinf and xsup, respectively, whereas x = midrad(xmid, xrad) checks xrad $\geq$ 0 and defines an interval x containing [xmid − xrad, xmid + xrad]. In this case x may be a superset of [xmid − xrad, xmid + xrad] because real intervals are stored in INTLAB by their left and right bound. The type cast and initializations of interval quantities may be applied to vector and matrix quantities xinf, xsup, xmid and xrad as well.

If at least one operand in a Matlab/INTLAB operation is of type intval, then the corresponding interval operation is executed, possibly using the natural embedding for the other operand. For example, after line 5 in Algorithm 3.1 (LssErrBndDirRdg) the statement r = R * (A * intval(xs) − b) would calculate a rigorous inclusion $\mathtt{r} \in \mathbb{IF}^n$ of $\mathtt{R} \cdot (\mathtt{A} \cdot \mathtt{xs} - \mathtt{b})$. This is not necessarily true for rs = intval(R) * (A * xs − b) because the residual res = A * xs − b is calculated in floating-point with $\mathtt{res} \in \mathbb{F}^n$. For all INTLAB operators the rounding mode before and after execution is the same.

As a first example we rewrite Algorithm 3.3 (Dot2DirRdg) in INTLAB using interval operations:

Algorithm 4.1. Matrix product $\mathtt{A} \cdot \mathtt{B}$ for $\mathtt{A} \in \mathbb{F}^{m\times k}$, $\mathtt{B} \in \mathbb{F}^{k\times n}$ "as if" accumulated in twice the working precision with approximate result $\mathtt{C} \in \mathbb{F}^{m\times n}$ for Incl = 0, and rigorous inclusion $\mathtt{C} \in \mathbb{IF}^{m\times n}$ for Incl = 1.

```
function C = Dot2(A,B,Incl)
  if nargin==2, Incl=0; end              % no error term if "Incl" not specified
  [p,e] = TwoProduct(A(:,1),B(1,:));     % error-free transformation of first product
  k = size(A,2);                         % inner dimension
  for i=2:k                              % matrix product by rank-1 updates
    [h,r] = TwoProduct(A(:,i),B(i,:));   % error-free transformation of i-th product
    [p,q] = TwoSum(p,h);                 % error-free transformation of accumulated sum
    if Incl                              % rigorous error bounds to be computed
      t = q + intval(r);                 % inclusion of sum of errors
    else                                 % approximation to be computed
      t = q + r;                         % approximation of sum of errors
    end
    e = e + t;                           % approximate accumulation of errors
  end
  if Incl, e=e+midrad(0,max(1.5*k*eps,1)*realmin); end   % cover possible underflow
  C = p + e;                             % extra-precise result
```

In view of Dot2DirRdg the algorithm seems self-explaining, only shorter and simpler. Note that only t, e and C are interval quantities for Incl = 1; for Incl = 0 only floating-point quantities occur in the algorithm. Also note that the multiplications 1.5 ∗ k ∗ eps ∗ realmin do not cause a rounding error and may safely be executed in rounding to nearest.

Next we rewrite Algorithm 3.1 (LssErrBndDirRdg) for linear systems the data of which may be afflicted with tolerances. In this case the set of all $A^{-1}b$ for A and b within the tolerances is included together with the proof of non-singularity of all such matrices A.

Algorithm 4.2. Rigorous inclusion $\mathtt{x} \in \mathbb{IF}^n$ of the solution of the linear systems $Ax = b$. If the matrix and/or right hand side are afflicted with tolerances, i.e. $\mathtt{A} \in \mathbb{IF}^{n\times n}$ and/or $\mathtt{b} \in \mathbb{IF}^n$, then the inclusion covers all linear systems within the given data (together with the proof of non-singularity of all matrices in A).

```
 1  function x = LssErrBnd(A,b,Illco)
 2    setround(0)                                % rounding to nearest
 3    x = intval(NaN(size(b))); n = size(A,2);   % initialization
 4    R = inv(mid(A));                           % approximate inverse of mid(A)
 5    xs = ResidIter(mid(A),mid(b),R);           % approximate solution of midpoint system
 6    tol = ( any(rad(A(:))>0) ) || ( any(rad(b)>0) ); % check tolerances
 7    if tol                                     % input data with tolerances
 8      c = mag(R*(A*intval(xs)-b));             % upper bound of |R(A*xs-b)|
 9    else                                       % extra-precise accumulation
10      c = mag(R*Dot2([A b],[xs;-1],1));        % upper bound of |R(A*xs-b)|
11    end
12    C = R*intval(A);                           % inclusion of RA
13    d = diag(C.inf);                           % lower bound of diag(RA)
14    if all(d>0)                                % C may be H-matrix
15      E = mag(C); E(1:n+1:n^2) = 0;            % <C>=D-E  with D,E>=0, D:=diag(d)
16      [u,v] = MVector(E,d);                    % vector iteration to verify M-property
17      if all(u>0) && all(v>0)                  % C proved to be H-matrix
18        setround(1)                            % rounding to upwards
19        w = max(E./(u*d'));                    % upper bound (u*w)_ik >= (E*D^-1)_ik
20        dinv = 1./d;                           % upper bound of diag(D^-1)
21        err = dinv.*c + v*(w*c);               % error bound for xs
22        N = inf;                               % initialize constant
23        for iter=1:15                          % at most 15 improvements of error bound
24          err = min(err, dinv.*(c + E*err) ); % improved error bound
25          Nold = N; N = max(err);              % norm of correction
26          if N >= 0.99*Nold, break, end        % stop iteration if no improvement
27          if max(abs(err)./abs(xs))<=eps, break, end  % already maximally accurate
28        end
29        x = xs + midrad(0,err);                % final inclusion
30        setround(0)                            % rounding to nearest
31        return
32      end
33    end
34    if ( nargin>2 ) && Illco && (~tol)         % solution by phase II
```

Table 4.1
Definition of mid, rad and mag for non-interval and interval input x.

	$x \in \mathbb{F}$	$x \in \mathbb{IF}$
mid(x)	x	$mx := mid(x) \in \mathbb{F}$
rad(x)	0	$rx := rad(x) \in \mathbb{F}$ } such that $x \subseteq [mx - rx, mx + rx]$
mag(x)	$\|x\|$	$\max\{\|x\| : x \in x\} \in \mathbb{F}$

```
35      x = LssErrBnd(Dot2(R,A,1),Dot2(R,b,1),0);
36    end
37    setround(0)                                  % rounding to nearest
```

In the remainder of this section Algorithm LssErrBnd is analyzed and proved to be correct for the call

$$x = \texttt{LssErrBnd}(A, b, \texttt{Illco}) \quad \text{with } \texttt{Illco} = 0, \tag{4.5}$$

in particular for input data with tolerances. Despite the type cast intval the functions mid, rad and mag are used. For a floating-point number $x \in \mathbb{F}$ or interval $x \in \mathbb{IF}$ they are defined as in Table 4.1.

Note that the quantities $mx := mid(x)$ and $rx := rad(x)$ are computed such that $x \subseteq [mx - rx, mx + rx]$; they are not necessarily equal to the true midpoint and radius, respectively, because those do not need to be in $\mathbb{F}$. All definitions apply to vector and matrix quantities entrywise.

First assume $A \in \mathbb{F}^{n\times n}$ and $b \in \mathbb{F}^n$ are given. Then mathematically there is no difference to Algorithm 3.1 (LssErrBndDirRdg): By definition $mid(A) = A$, $mid(b) = b$, and $tol = 0$, so that R and xs are identical, and c as computed in line 10 as well. No matter how the inclusion C of $R \cdot A$ in line 12 is computed, all assertions remain valid as long as $R \cdot A \subseteq C$ (in fact, INTLAB computes C in line 10 exactly as in Algorithm LssErrBndDirRdg). The rest of Algorithm LssErrBnd up to line 33 computes the same quantities as Algorithm LssErrBndDirRdg, only the final result is stored in an interval vector $x \in \mathbb{F}^n$ rather than in an approximation with separate error term. The simple check in line 27 for already achieved maximum accuracy is fast and proves to be useful. Since $Illco = 0$, line 35 is not executed.

Next assume $A \in \mathbb{IF}^{n\times n}$ or $b \in \mathbb{IF}^n$, or both. In lines 4 and 5 an approximate inverse of mid(A) and an approximate solution xs of the midpoint linear system is computed. Mathematically the concrete values are irrelevant, all assertions are valid for any $R \in \mathbb{F}^{n\times n}$ and $xs \in \mathbb{F}^n$. The extra-precise evaluation of the correction $R(A\tilde{x} - b)$ does not gain accuracy because interval quantities are involved in A or b. This would change if only a few entries in A or b have nonzero diameter, however, this seems untypical. Therefore an upper bound $c \in \mathbb{F}^n$ of $|R \cdot (A \cdot xs - b)|$ is computed by standard interval arithmetic in line 8 using mag as in Table 4.1.

Now let fixed but arbitrary $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ with $A \in A$ and $b \in b$ be given (in case A or b is a non-interval quantity, we use the natural embedding). Note that A and b are arbitrary real quantities within the interval bounds. Then the inclusion property (4.3) implies $C := R \cdot A \in C$ for C as computed in line 12.

Let $C = D - E$ the splitting of C into diagonal and off-diagonal parts. The left, i.e. lower bound of the diagonal of C is stored in d, and $d > 0$ implies $D \geq diag(d) > 0$. Moreover, $C \in C$ implies $|C| \leq mag(C)$, so that $|E| \leq E$ after execution of line 15. When arriving in line 18, vectors u, $v > 0$ have been computed by MVector(E,d) in line 16 satisfying $(diag(d) - E) \cdot v \geq u > 0$. Using $D \geq 0$ this implies

$$\langle C \rangle \cdot v = (D - |E|) \cdot v \geq (diag(d) - E) \cdot v > 0,$$

so that $\langle C \rangle$ is an M-matrix. Next an upper bound of $\langle diag(d) - E \rangle^{-1}$ is computed as in Algorithm 3.1 (LssErrBndDirRdg). Hence a standard result from the theory of M-matrices (Theorem 2.5.4 in [7]) implies

$$C^{-1} \leq \langle C \rangle^{-1} \leq \langle diag(d) - E \rangle^{-1},$$

so that $|R \cdot (A \cdot xs - b)| \leq c$ and the rounding to upwards as set in line 18 prove that A is non-singular and that x is an inclusion of $A^{-1}b$. Since A and b were chosen arbitrarily within A and b, this proves the following result.

Theorem 4.3. *Let a matrix $A \in \mathbb{F}^{n\times n}$ and a right hand side $b \in \mathbb{F}^n$ be given, and let $x \in \mathbb{F}^n$ be the result of Algorithm 4.2 (LssErrBnd) computed by the call $x = \texttt{LssErrBnd}(A, b, 0)$. Then*

$$A \text{ is non-singular} \quad \text{and} \quad A^{-1}b \in x \tag{4.6}$$

for $A = A$ and $b = b$. If $A \in \mathbb{IF}^{n\times n}$ and/or $b \in \mathbb{IF}^n$, then (4.6) is true for all $A \in A$ and $b \in b$. The computational effort for $A \in \mathbb{F}^{n\times n}$ is $3n^3 + \mathcal{O}(n^2)$ operations, and for $A \in \mathbb{IF}^{n\times n}$ it is $4n^3 + \mathcal{O}(n^2)$ operations.

5. Alternative approaches to calculate rigorous error bounds for linear systems

Given $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$, we discuss in this section alternative approaches to prove A to be non-singular and to calculate rigorous error bounds for $A^{-1}b$.

5.1. The Krawczyk operator

Let $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ be given together with some approximate inverse R of A, an approximate solution $\widetilde{x}$, and some interval vector $\mathbf{x} \in \mathbb{IR}^n$. Suppose

$$\widetilde{x} + R(b - A\widetilde{x}) + (I - RA)(\mathbf{x} - \widetilde{x}) \subseteq \mathbf{x}, \tag{5.1}$$

where for simplicity of exposition all operations are power set operations. Then Krawczyk [17] showed that, if $\|I - RA\| < 1$ for some norm, A is non-singular and $A^{-1}b \in \mathbf{x}$. The left hand side of (5.1) is called the Krawczyk operator. This has been simplified and improved by Rump [18] as follows. Suppose

$$R(b - A\widetilde{x}) + (I - RA)\mathbf{x} \subseteq \mathrm{int}(\mathbf{x}), \tag{5.2}$$

where $\mathrm{int}(\cdot)$ denotes the topological interior. Then A is non-singular and $A^{-1}b \in \widetilde{x} + \mathbf{x}$. No assumption on a norm of $I - RA$ is necessary. It is often superior to compute an inclusion of the error with respect to the approximation $\widetilde{x}$.

Moreover, Krawczyk did not provide a method how to find a suitable $\mathbf{x}$ except the obvious choice $\tilde{x} \pm \varepsilon$. In [18] we introduced and analyzed the "so-called" epsilon-inflation to construct a suitable $\mathbf{x}$. The method is implemented as algorithm `verifylss(A,b)` in INTLAB. Although the method is more than 30 years old, it seems to be the state of the art for computing rigorous inclusions for general dense matrices [19].

A successful application of (5.2) requires the necessary condition $\varrho(|I - RA|) < 1$ to hold true. Under reasonable assumptions and using the epsilon-inflation, this condition is also sufficient [18], i.e. the interval-iteration ends successfully. For small enough positive δ and $E = ee^T$ this condition implies $r := \varrho(\delta E + |I - RA|) < 1$, and by Perron–Frobenius theory the existence of some $0 < u \in \mathbb{R}^n$ with

$$\big(\delta E + |I - RA|\big)u = ru \quad \text{with } r < 1. \tag{5.3}$$

Hence, adapting the proof of Proposition 3.7.2 in [15],

$$ru > |I - RA|u \geq (\langle I\rangle - \langle RA\rangle)u = u - \langle RA\rangle u, \tag{5.4}$$

so that $\langle RA\rangle u > (1-r)u > 0$ proves RA to be an H-matrix. Conversely, such as for $RA = -I$, RA may be an H-matrix without $\varrho(|I - RA|) < 1$ being true.

As a consequence, Theorem 2.1 is always applicable whenever the Krawczyk operator or (5.2) is. If R is scaled so that all diagonal elements of RA are equal to 1, then $D = I$ and $|I - RA| = E$, so that $\varrho(|I - RA|) < 1$ is equivalent to RA being an H-matrix, and (5.2) and Algorithm 4.2 (`LssErrBnd`) have the same scope of applicability.

The mentioned interval-iteration has the same effect as a Perron iteration. Therefore we did not find a case where Theorem 2.1 succeeds to compute an inclusion but (5.2) fails, see the computational results presented in Section 8.

5.2. The theorem by Hansen/Bliek/Rohn/Ning/Kearfott/Neumaier

The theoretical background for linear systems with tolerances, which we used implicitly in the previous section, is as follows. The mignitude and magnitude of a (scalar) interval $\mathbf{x} \in \mathbb{IR}$ is defined by

$$\mathrm{mig}(\mathbf{x}) := \min\{|x| : x \in \mathbf{x}\} \in \mathbb{R} \quad \text{and} \quad \mathrm{mag}(\mathbf{x}) := \max\{|x| : x \in \mathbf{x}\} \in \mathbb{R}. \tag{5.5}$$

The definition, like for midpoint and radius, extends to interval vectors and matrices entrywise. For the magnitude we use also the intuitive notation $|\mathbf{x}|$. For an interval matrix $\mathbf{A} \in \mathbb{IR}^{n\times n}$, Ostrowski's comparison matrix $\langle\mathbf{A}\rangle \in \mathbb{R}^{n\times n}$ is, similar to (1.1), defined by Neumaier [15]

$$\langle\mathbf{A}\rangle_{ii} := \mathrm{mig}(\mathbf{A}_{ii}) \quad \text{and} \quad \langle\mathbf{A}\rangle_{ij} := -\mathrm{mag}(\mathbf{A}_{ij}) \quad \text{for } i \neq j. \tag{5.6}$$

If $C := \langle\mathbf{A}\rangle \in \mathbb{R}^{n\times n}$ is an M-matrix, then $\mathbf{A}$ is called an H-matrix. In that case every $A \in \mathbf{A}$ is non-singular, and $|A^{-1}| \leq C^{-1}$ (cf. [15]). The hull of a set $X \subseteq \mathbb{R}^n$ is defined to be the smallest interval vector containing X, i.e.

$$\mathrm{hull}(X) := \bigcap\{\mathbf{x} \in \mathbb{IR}^n : X \subseteq \mathbf{x}\}. \tag{5.7}$$

We use interval operations as defined in (4.2). For given $\mathbf{A} \in \mathbb{IR}^{n\times n}$, $\mathbf{b} \in \mathbb{IR}^n$ define

$$\Sigma(\mathbf{A}, \mathbf{b}) := \{x \in \mathbb{R}^n : \exists A \in \mathbf{A}, \exists b \in \mathbf{b}, Ax = b\}. \tag{5.8}$$

Theorem 4.3 shows that Algorithm 4.2 (`LssErrBnd`) computes some $\mathbf{x}$ with $\mathrm{hull}\big(\Sigma(\mathbf{A}, \mathbf{b})\big) \subseteq \mathbf{x}$. This inclusion, however, may be an overestimation due to data dependencies. In Section 7 we will discuss how to measure this. Much work has been invested to reduce this overestimation [15]. It was hoped that a clue would be the theorem to be discussed in this subsection: For certain matrices the *exact hull* of the solution set $\Sigma(\mathbf{A}, \mathbf{b})$ can be computed efficiently. In general, the problem is NP-hard.

For given $\mathbf{A} \in \mathbb{IR}^{n\times n}$ one may use, as in Algorithm 4.2 (`LssErrBnd`), an approximate inverse R of the midpoint matrix as a preconditioner. Define $\mathbf{C} := R \cdot \mathbf{A} \in \mathbb{IR}^{n\times n}$ and $C := \langle\mathbf{C}\rangle \in \mathbb{R}^{n\times n}$. If the tolerances in $\mathbf{A}$ are not too wide and the matrices in $\mathbf{A}$ not too ill-conditioned, it is not unlikely that $\mathbf{C}$ is an H-matrix. In that case the following theorem is applicable. As the number of authors suggests, the theorem has a history [20–23,6,24]. We state the version by Neumaier [6,24], who presented a short and nice proof. Over there also historical remarks appear.

Theorem 5.1. *Let $\mathbf{C} \in \mathbb{IR}^{n\times n}$ be an H-matrix, $\mathbf{c} \in \mathbb{IR}^n$ a right hand side,*

$$r = \langle \mathbf{C}\rangle^{-1}|\mathbf{c}|, \qquad \delta_i = (\langle \mathbf{C}\rangle^{-1})_{ii}, \qquad \alpha_i = \langle \mathbf{C}\rangle_{ii} - 1/\delta_i, \qquad \beta_i = r_i/\delta_i - |\mathbf{c}_i|. \tag{5.9}$$

Then hull$\big(\Sigma(\mathbf{C}, \mathbf{c})\big)$ is contained in the vector $\mathbf{x} \in \mathbb{IR}^n$ with the components

$$\mathbf{x}_i = \frac{\mathbf{c}_i + [-\beta_i, \beta_i]}{\mathbf{C}_{ii} + [-\alpha_i, \alpha_i]}. \tag{5.10}$$

Moreover, if the midpoint of $\mathbf{C}$ is diagonal, then hull$\big(\Sigma(\mathbf{C}, \mathbf{c})\big) = \mathbf{x}$.

The astonishing part of this theorem is the last sentence: if all off-diagonal entries of $\mathbf{C}$ are centered around zero, then $\mathbf{x}$ is equal to the true hull of the solution set hull$\big(\Sigma(\mathbf{A}, \mathbf{b})\big)$.

For the practical application of Theorem 5.1, the quantities in (5.9) have to be estimated. Following this we discuss and improve Neumaier's approach [6]. The critical part is an upper bound for the inverse of $\langle \mathbf{C}\rangle$, where for the diagonal δ a lower bound is required as well.

Since $\langle \mathbf{C}\rangle \in \mathbb{R}^{n\times n}$, we split $\langle \mathbf{C}\rangle = D - E$ into diagonal and off-diagonal parts as in (2.4). Then the estimate (2.7) of $\langle \mathbf{C}\rangle^{-1}$ requires a positive vector $v \in \mathbb{R}^n$ with $u = \langle \mathbf{C}\rangle v > 0$. Neumaier [6] proposes to compute an approximate inverse B of the midpoint of $\langle \mathbf{C}\rangle$ and uses the natural choice $v := Be$ with $e := (1, \ldots, 1)^T$, for which one can expect that $\langle \mathbf{C}\rangle v$ is not too far from e. Then he computes a nonnegative vector w similar to in (2.6) satisfying $I - \langle \mathbf{C}\rangle B \le uw^T$. By $uw^T \ge 0$ also $\langle \mathbf{C}\rangle B - I \le uw^T$, and multiplying from the left by $\langle \mathbf{C}\rangle^{-1} \ge 0$ and using $u = \langle \mathbf{C}\rangle v$ he concludes

$$B - vw^T \le \langle \mathbf{C}\rangle^{-1} \le B + vw^T. \tag{5.11}$$

This is used by Neumaier [6,24] to estimate α and β in (5.10).

Neumaier [25] was not interested in giving a most efficient algorithm but in a nice and simple proof of Theorem 5.1. Choosing B to be an approximate inverse of (the midpoint of) $\langle \mathbf{C}\rangle$ can be interpreted as an extra preconditioning[1] which requires extra matrix multiplications, so that the total computational effort to compute bounds for $A^{-1}b$ is more than doubled.

Another common choice for v is $v = e$ resulting in $B = I$. However this may result in failure to verify $\langle \mathbf{C}\rangle$ to be an M-matrix as has been discussed earlier.

But the choices of B and v are independent: $B \in \mathbb{R}^{n\times n}$ can be chosen arbitrarily, only $u = \langle \mathbf{C}\rangle v > 0$ must be satisfied for some positive v. As has been discussed in Section 2.1, the Perron vector computed by Algorithm 3.2 (`MVector`) is the best choice for v, and a simple preconditioning matrix B for $\langle \mathbf{C}\rangle$ suffices.

Theorem 5.2. *Let $\mathbf{C} \in \mathbb{IR}^{n\times n}$ and $\mathbf{c} \in \mathbb{IR}^n$ together with a positive vector v with $u := \langle \mathbf{C}\rangle v > 0$ be given. Let $D-E = \langle \mathbf{C}\rangle \in \mathbb{R}^{n\times n}$ be the splitting of $\langle \mathbf{C}\rangle$ into diagonal and off-diagonal parts, and let $w \in \mathbb{R}^n$ be such that $I - \langle \mathbf{C}\rangle B \le uw^T$. Then $\mathbf{C}$ is an H-matrix and D is invertible. Abbreviating $d_i := D_{ii} = \langle \mathbf{C}\rangle_{ii}$, the quantities $\alpha, \beta \in \mathbb{R}^n$ defined in Theorem 5.1 satisfy*

$$\alpha_i \le \frac{d_i}{1 + 1/((Ev)w^T)_{ii}} \quad \text{and} \quad \beta_i \le w^T|\mathbf{c}| \cdot d_i v_i, \tag{5.12}$$

and hull$\big(\Sigma(\mathbf{C}, \mathbf{c})\big)$ is contained in the vector $\mathbf{x} \in \mathbb{IR}^n$ defined by (5.10). Moreover,

$$\beta_i \le \frac{\big[E(D^{-1} + vw^T)|\mathbf{c}|\big]_i - N_i|\mathbf{c}|_i}{1 + N_i}, \tag{5.13}$$

where $N_i := \big[(ED^{-1})^2\big]_{ii}$.

Remark 1. Note that only the diagonal of $(ED^{-1})^2$ is needed for N_i, so the computational effort is $\mathcal{O}(n^2)$.

Remark 2. If $\mathbf{C} := R \cdot \mathbf{A}$ for some given interval matrix $\mathbf{A} \in \mathbb{IR}^{n\times n}$ and an approximate midpoint inverse R as preconditioner, then the off-diagonal elements E in $\langle \mathbf{C}\rangle$ can be expected to be small.

Proof. First we proceed as in (2.5) and (2.6) to see

$$D^{-1} \le \langle \mathbf{C}\rangle^{-1} \le D^{-1} + vw^T, \tag{5.14}$$

where the lower bound follows by $(D - E)^{-1} = \langle \mathbf{C}\rangle^{-1} = D^{-1}(I - ED^{-1})^{-1}$, the Neumann expansion and $D, E \ge 0$. As has been mentioned in Section 2, the expansion $D^{-1}(I - ED^{-1})^{-1} = D^{-1}\big(I + ED^{-1}(I - ED^{-1})^{-1}\big)$ yields

$$\langle \mathbf{C}\rangle^{-1} \le D^{-1}\big(I + E(D^{-1} + vw^T)\big). \tag{5.15}$$

[1] Note that a right inverse is necessary or `B = inv(C')'` in Matlab notation.

Expanding the Neumann series one more term and observing that the diagonal of E is zero, we obtain

$$\operatorname{diag}\big(D^{-1}(I+(ED^{-1})^2)\big) \le \operatorname{diag}\big(\langle\mathbf{C}\rangle^{-1}\big) \le \operatorname{diag}\big(D^{-1}(I+(Ev)w^T)\big), \tag{5.16}$$

and using (5.16) and $\delta_i = (\langle\mathbf{C}\rangle^{-1})_{ii}$ yields

$$\alpha_i = d_i - 1/\delta_i \le d_i\left(1 - \frac{1}{1+((Ev)w^T)_{ii}}\right) = \frac{d_i}{1+1/((Ev)w^T)_{ii}}. \tag{5.17}$$

The upper bound for β uses $\delta_i \ge d_i^{-1}$ as by (5.14) to conclude

$$\beta_i = \big(\langle\mathbf{C}\rangle^{-1}|\mathbf{c}|\big)_i/\delta_i - |\mathbf{c}|_i \le \frac{d_i^{-1}|\mathbf{c}|_i + (vw^T|\mathbf{c}|)_i}{d_i^{-1}} - |\mathbf{c}|_i, \tag{5.18}$$

so that (5.12) follows. A little computation using (5.16) shows (5.13) and completes the proof. □

Unfortunately, according to our numerical evidence, these improved estimates on the quantities used in Theorem 5.1 also do not lead to better bounds than our methods presented in the previous section, see the computational results in Section 8. This seems surprising because the bound given in Theorem 5.1 is provably optimal under reasonable assumptions.

However, the optimality is only true if the entries in $\mathbf{C}$ are independent, which is not true for $\mathbf{C} = R \cdot \mathbf{A}$. There is hardly an efficient remedy for this because the computation of the hull of the solution set $\Sigma(\mathbf{A}, \mathbf{b})$, even of an approximation to a certain degree, is an NP-hard problem (see [26–29]).

5.3. Ogita's method

Let $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ together with an approximate inverse R of A be given. In [30] Ogita et al. assumes $\|I - RA\| < 1$ and uses the standard estimate

$$\|A^{-1}\| = \left\|\big(I-(I-RA)\big)^{-1}R\right\| \le \frac{\|R\|}{1-\|I-RA\|}$$

to conclude[2]

$$\|\widetilde{x} - A^{-1}b\| = \|\tilde{z} - A^{-1}(A\tilde{z}-r)\| \le \|\tilde{z}\| + \frac{\|R(A\tilde{z}-r)\|}{1-\|I-RA\|}, \tag{5.19}$$

where $r = A\widetilde{x} - b$ is the residual of some approximate solution $\widetilde{x}$ of the original system $Ax = b$, and $\tilde{z}$ is an approximate solution of the residual equation $Az = r$. The authors use $\|\cdot\|_\infty$. Since $\|I - RA\|_\infty = \| \, |I - RA|e \, \|$, the scope of applicability could be increased to that of `LssErrBnd` by choosing the Perron vector v instead of e to verify $\|D_v^{-1}|I - RA|D_v\| < 1$.

The residual is computed using some extra-precise dot product accumulation, so there is some similarity to (4.1) in Part I. The idea is that the residual r is small, so that the residual $A\tilde{z} - r$ of the residual equation $Az = r$ is extra small.

The verification in [30] is carried out in rounding to nearest and estimation of rounding errors. In the Matlab code we received by Ogita he uses interval operations and INTLAB, so this method may safely be compared with our algorithms. Computational results are presented in Section 8.

Moreover, a method for inverse monotone matrices, i.e. A is non-singular and $A^{-1} \ge 0$, is presented in [31]. Given a monotone matrix C and an approximate solution $\tilde{y}$ of $Cy = e$ with $e = (1, \ldots, 1)^T$, they use

$$\|A^{-1}\|_\infty = \|A^{-1}e\|_\infty = \|A^{-1}(e - Ay) + y\|_\infty \le \|A^{-1}\|_\infty \|Ay - e\|_\infty + \|y\|_\infty$$

to conclude

$$\|A^{-1}\|_\infty \le \frac{\|y\|_\infty}{1 - \|Ay - e\|_\infty}. \tag{5.20}$$

If inverse monotonicity is known *a priori* from some given application, then (5.20) gives a fast estimate of $\|A^{-1}\|_\infty$. This method is also applicable to the matrix $C = RA$ using $C^{-1} \le \langle C\rangle^{-1}$ after verifying inverse monotonicity of $\langle C\rangle$ by $\langle C\rangle v > 0$ for positive v. Note, however, that (2.7) yields a componentwise estimate of C^{-1} in contrast to the normwise estimate (5.20).

Numerical tests suggest that usually (2.7) is better than (5.20), but sometimes worse. However, we are interested in estimating $|C^{-1}c|$ for c denoting the correction $R(A\widetilde{x} - b)$. In (2.8) we could estimate the absolute value of this product, whereas using (5.20) only $\|C^{-1}\|_\infty \|c\|_\infty$ is estimated. The latter is always worse than (2.8).

5.4. Nguyen's method

Let $A \in \mathbb{R}^{n\times n}$ together with an approximate inverse R of A be given. Then Nguyen and Revol [32] and Nguyen [33] discuss another method based on verifying that RA is an H-matrix. Rather than the obvious choice $v = e := (1, \ldots, 1)^T$ for

[2] In [30] the numerator in the last term in (5.19) was given as $\|R\| \cdot \|A\tilde{z} - r\|$, which can obviously be improved into $\|R(A\tilde{z} - r)\|$; in the numerical tests, (5.19) was used.

checking $\langle RA\rangle v > 0$ he also uses an iteration on v. His iteration is based on some Jacobi-iteration. In our notation he uses $D - E = C := \langle RA\rangle$.

Algorithm 5.3. Nguyen's vector iteration to find v with $\langle RA\rangle v > 0$.

$v = e$
for $i = 1 : i_{\max}$
 $v = D^{-1}(e + Ev)$
 $u = Cv$
 if $u > 0$, break, end if
end for.

This iteration is almost as good as the Perron iteration presented in Section 2.1; however, it happens for ill-conditioned matrices that it fails to verify RA to be an H-matrix, whereas Algorithm 3.2 (`MVector`) based on the Perron iteration succeeds, see Section 8. This confirms the analysis in Section 2.1.

Suppose $v > 0$ is given with $u := \langle RA\rangle v > 0$. Then for $b, \widetilde{x} \in \mathbb{R}^n$, $r := A\widetilde{x} - b$, $c := R(A\widetilde{x} - b)$ and $D_u := \mathrm{diag}(u)$, Proposition 4.1.9 in [15] gives

$$|\widetilde{x} - A^{-1}b| \le \|D_u^{-1}c\|_\infty v. \tag{5.21}$$

This is true by observing $C^{-1} = \langle RA\rangle^{-1} \ge 0$ and $|\widetilde{x} - A^{-1}b| = |C^{-1}c| \le C^{-1} \cdot \|D_u^{-1}c\|_\infty u = \|D_u^{-1}c\|_\infty v$. Nguyen uses this to obtain an error bound similar to (2.8). But rather than improving the initial value $\widetilde{x} := Rb$ by an *a priori* residual iteration, he combines this with an inclusion by (5.21) as follows:

Compute an inclusion $\mathbf{r}$ of $A\widetilde{x} - b$ using extra-precise dot product accumulation
Compute an inclusion of $\mathbf{z}$ of $R\mathbf{r}$
Initial error bound $e := \|D_u^{-1}\mathbf{r}\|_\infty v$
While (not converged)
 Apply five Gauss–Seidel iterations
 If accurate enough, break
 Update $\widetilde{x}$ by $\widetilde{x} + e$
 Update e by $e - \mathrm{mid}(e)$
 Recompute $\mathbf{r}$ and $\mathbf{z}$
End while.

Nguyen discusses three basic ideas to speed up the inclusion method. First, the iteration can be stopped if the already achieved inclusion is sufficiently accurate. To decide this he uses two stopping criteria, namely

$$-\log_2\left(\max_i \frac{\mathrm{rad}(e)}{|\widetilde{x}_i|}\right) \ge 52 \quad \text{or} \quad \max_i \frac{|\mathrm{rad}(e'_i) - \mathrm{rad}(e_i)|}{|\widetilde{x}_i|} < \mathbf{u},$$

where e' denotes the error bound in the previous iteration. This trick may avoid unnecessary iterations by utilizing the knowledge of an inclusion. Since Gauss–Seidel iterations suffer significantly from interpretation overhead, he uses a Jacobi-iteration instead.

Second, rather than improving an approximate solution $\widetilde{x}$ *a priori* by a (floating-point) residual iteration and then to start the verification process, Nguyen uses an initial approximation and combines the verification process with the interval residual iteration process. If the computed inclusion is not accurate enough, he uses the midpoint of the interval correction $\mathbf{z}$ to improve the approximate solution $\widetilde{x}$. Then he proceeds to compute the next inclusion.

This trick is working fine if the matrix A is not too ill-conditioned; otherwise, however, the inclusion $\mathbf{r}$ of the residual is multiplied by R. Although $\mathbf{r}$ is very narrow due to the extra-precise evaluation, the (small) radii are amplified by the ill-conditioned matrix R resulting in relatively wide intervals of the product $\mathbf{z}$. Hence the midpoint of $\mathbf{z}$, which is an inclusion of the correction $R(A\widetilde{x} - b)$, is often not as good a correction for $\widetilde{x}$ as $R(A\widetilde{x} - b)$ computed in floating-point. As a consequence, more residual and/or Gauss–Seidel or Jacobi iterations are necessary, see the computational results in Section 8.

Third, he uses that $\langle RA\rangle$ is close to the identity matrix and proposes a relaxed version, where certain interval operations are replaced by floating-point operations in rounding to upwards. We received the Matlab code from Nguyen using INTLAB for his method, so this method may also safely be compared with our algorithms. Computational results are presented in Section 8.

5.5. Sparse matrices

We briefly mention yet another approach [34] for symmetric positive definite matrices. Let two $n \times n$-matrices $A^T = A$ and G be given and assume $\|A - \alpha I - G^T G\|_2 \le \tau$ for some positive α and τ. Then for all i,

$$\lambda_i(A) = \alpha + \lambda_i(A - \alpha I) \ge \alpha + \lambda_i(G^T G) - \tau \ge \alpha - \tau, \tag{5.22}$$

so that $\alpha > \tau$ implies that A is symmetric positive definite and $\sigma_{\min}(A) \geq \alpha - \tau$. Note that A is not assumed *a priori* to be positive definite. An error bound for an approximate solution $\widetilde{x}$ follows by

$$\|\widetilde{x} - A^{-1}b\|_2 \leq \|A^{-1}\|_2 \cdot \|A\widetilde{x} - b\|_2 \leq (\alpha - \tau)^{-1}\|A\widetilde{x} - b\|_2. \tag{5.23}$$

Apparently this is still the only efficient method to compute rigorous error bounds for sparse matrices; an efficient method for general or even symmetric indefinite matrices is still to be found. We mention this method for completeness; it is designed for sparse matrices and not competitive for full matrices with the methods presented in this paper. On the other hand, the latter use an approximate inverse and are thus not suited for sparse matrices.

6. Inclusion of extremely ill-conditioned linear systems

In Part I of this paper we presented a method to compute verified error bounds for the solution of a linear system $Ax = b$ with an extremely ill-conditioned matrix, i.e. $\mathrm{cond}(A) \gtrsim \mathbf{u}^{-1}$. We gave Algorithm 4.20 (`LssIllcoErrBndNear`) in Part I in executable code using only rounding to nearest.

If directed rounding is available and the possibility to compute an inclusion of the solution set $\Sigma(\mathbf{A}, \mathbf{b})$ of a linear system the data of which is afflicted with tolerances, then we may proceed in a much simpler way.

Let $A \in \mathbb{F}^{n\times n}$ and $b \in \mathbb{F}^n$ be given. For $R \in \mathbb{F}^{n\times n}$ let $\mathbf{C} \in \mathbb{IF}^{n\times n}$ and $\mathbf{c} \in \mathbb{IF}^n$ be such that

$$RA \in \mathbf{C} \quad \text{and} \quad Rb \in \mathbf{c}. \tag{6.1}$$

If one of our algorithms applied to $\mathbf{C}$ and $\mathbf{c}$ succeeds to compute an inclusion $\mathbf{x}$ of $\Sigma(\mathbf{C}, \mathbf{c})$, then we know that (1) all matrices $C \in \mathbf{C}$ are non-singular, and (2) that $C^{-1}c \in \mathbf{x}$ is true for all $C \in \mathbf{C}$ and $c \in \mathbf{c}$. This is in particular true for $C := RA \in \mathbf{C}$ and $c := Rb \in \mathbf{c}$, which implies that A (and R) is non-singular, and that

$$(RA)^{-1}Rb = A^{-1}b \in \mathbf{x}. \tag{6.2}$$

Thus we may apply Algorithm 4.2 (`LssErrBnd`) to an interval matrix and vector including RA and Rb, respectively. This is done in line 35 of Algorithm `LssErrBnd` by the call `LssErrBnd(A,b,1)`.

This approach is not applicable if the original matrix A is already afflicted with tolerances and the midpoint matrix, say, is extremely ill-conditioned. The weighted 2-norm distance to the nearest singular matrix is $\mathrm{cond}(A)$, so that for $\mathrm{cond}(A) \gtrsim \mathbf{u}^{-1}$ it is likely that small tolerances on A produce a singular matrix.

It is important to calculate RA and Rb using extra-precise dot products. Suppose an inclusion $\mathbf{C}$ of RA is computed in working precision. Then standard rounding error analysis tells us that the radius of $\mathbf{C}$ is of the order $\mathbf{u}|R \parallel A|$. Even for a perfect approximate inverse R this is of the order $\mathbf{u} \cdot \mathrm{cond}(A)$ or larger, so that for extremely ill-conditioned matrices $\mathbf{C}$ contains singular matrices. We obtain the following result.

Theorem 6.1. *Let a matrix* $\mathtt{A} \in \mathbb{F}^{n\times n}$ *and a right hand side* $\mathtt{b} \in \mathbb{F}^n$ *be given, and let* $\mathtt{x} \in \mathbb{F}^n$ *be the result of Algorithm 4.2 (`LssErrBnd`) computed by the call* $\mathtt{x} = \mathtt{LssErrBnd(A, b, 1)}$. *Then*

$$\mathtt{A} \text{ is non-singular} \quad \text{and} \quad \mathtt{A}^{-1}\mathtt{b} \in \mathtt{x}. \tag{6.3}$$

The computational effort is $7n^3 + \mathcal{O}(n^2)$ *operations plus* $4n^2$ *extra-precise dot product evaluations. Numerical evidence suggests that an inclusion is obtained up to cond*$(\mathtt{A}) \lesssim \mathbf{u}^{-2}$.

7. Inner inclusions for linear systems with tolerances

Consider the interval matrix $\mathbf{A}$ and interval vector $\mathbf{b}$

$$\mathbf{A} = \begin{pmatrix} 2 & [0.25, 1] \\ [1, 2] & [-3, -2.5] \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} [-1, 0.5] \\ [0, 0.5] \end{pmatrix}. \tag{7.1}$$

Application of Algorithm 4.2 (`LssErrBnd`) computes the inclusion (rounded to two decimals)

$$\mathbf{x} = \begin{pmatrix} [-0.56, 0.40] \\ [-0.57, 0.30] \end{pmatrix} \tag{7.2}$$

together with the proof that each matrix $A \in \mathbf{A}$ is non-singular. The question arises how conservative these error bounds are. An answer to that can be provided by a so-called inner inclusion. For given $\mathbf{A} \in \mathbb{IR}^{n\times n}$ and $\mathbf{b} \in \mathbb{IR}^n$, Algorithm `LssErrBnd` computes $\mathtt{x} \in \mathbb{IF}^n$ with $\Sigma(\mathbf{A}, \mathbf{b}) \subseteq \mathtt{x}$. An interval vector $\mathbf{y} \in \mathbb{R}^n$ is called "inner inclusion" if $\mathbf{y} \subseteq \mathrm{hull}(\Sigma(\mathbf{A}, \mathbf{b}))$. It follows that

$$\forall i \in \{1, \ldots, n\}\ \exists A_1, A_2 \in \mathbf{A}\ \exists b_1, b_2 \in \mathbf{b} : (A_1^{-1}b_1)_i \leq \left(\inf(\mathbf{y})\right)_i \quad \text{and} \quad \left(\sup(\mathbf{y})\right)_i \leq (A_2^{-1}b_2)_i. \tag{7.3}$$

This does not imply that for $y \in \mathbf{y}$ there exist $A \in \mathbf{A}$ and $b \in \mathbf{b}$ with $Ay = b$ (see Fig. 7.1). Nevertheless the quality of an (outer) inclusion can be judged to be close to optimal if there is not too much difference between this outer and an inner inclusion.

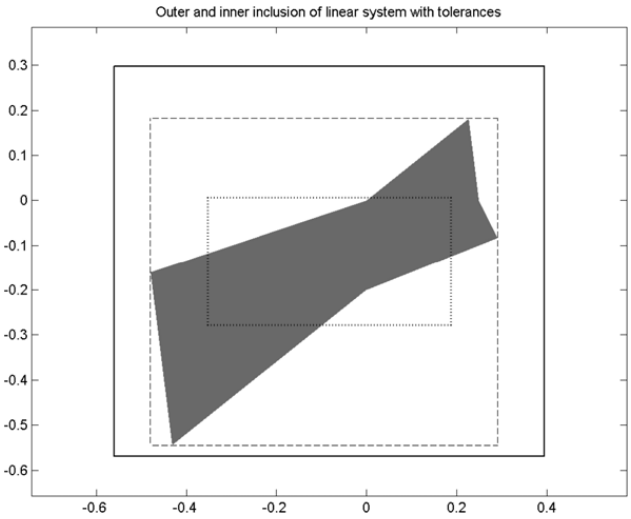

Fig. 7.1. Outer and inner inclusions for **A** and **b** as in (7.1).

Inner inclusions were introduced by Neumaier [35,36]. In [37] I showed how to compute inner inclusions with practically no additional computational effort. There is the drawback that the inner inclusion was often empty if not all entries of the matrix and right hand side were intervals of nonzero diameter. However, even if only one entry of the right hand side is an interval of nonzero diameter, the solution set $\Sigma(\mathbf{A}, \mathbf{b})$ has nonzero diameter in all entries except when A^{-1} has certain zero entries. If entries of **A** are intervals with nonzero diameter, then this is true except when all matrices A^{-1} for $A \in \mathbf{A}$ have certain zero entries. Both are highly unlikely. An example with only one entry of the matrix afflicted with a tolerance is given at the end of this section; results for only **b** being afflicted with tolerances are given in Section 8.

To show how to compute inner inclusions we use two representations of intervals simultaneously. For an interval $\mathbf{x} \in \mathbb{IR}$ we use both the infimum–supremum and midpoint-radius representation

$$\mathbf{x} = [x_{\inf}, x_{\sup}] = \{x \in \mathbb{R} : x_{\inf} \le x \le x_{\sup}\} = \langle mx, rx\rangle = \{x \in \mathbb{R} : |x - mx| \le rx\}. \tag{7.4}$$

The same notation applies to interval vectors and matrices using entrywise comparison and absolute value. Note that the notation implies $x_{\inf} \le x_{\sup}$ in case of $[x_{\inf}, x_{\sup}]$, and $rx \ge 0$ in case of $\langle mx, rx\rangle$. Define $\mathcal{S}_n := \{S \in \mathbb{R}^{n\times n} : |S| \le I\}$ and $\mathcal{E}_{mn} := \{E \in \mathbb{R}^{m\times n} : |E_{ij}| \le 1 \text{ for all } i, j\}$. We omit the indices when the dimension of $\mathcal{S}$ and $\mathcal{E}$ is clear from the context. Note that the signature matrices $\mathrm{diag}(\pm 1, \ldots, \pm 1)$ are the vertices of $\mathcal{S}$. Then

$$\begin{aligned} &\mathbf{x} = \langle mx, rx\rangle \in \mathbb{R}^n \Leftrightarrow \mathbf{x} = \{mx + S \cdot rx : \ S \in \mathcal{S}\} = \{mx + E \circ rx : \ E \in \mathcal{E}_{n1}\} \\ &\mathbf{A} = \langle mA, rA\rangle \in \mathbb{R}^{m\times n} \Leftrightarrow \mathbf{A} = \{mA + E \circ rA : \ E \in \mathcal{E}_{mn}\}, \end{aligned} \tag{7.5}$$

where $\circ$ denotes the Kronecker product (entrywise multiplication).

Lemma 7.1. *Let $\mathbf{y} = \langle my, ry\rangle \in \mathbb{IR}^{m\times n}$ be given. Suppose $\mu, \rho \in \mathbb{R}^{m\times n}$ satisfy*

$$my - ry \le \mu - \rho \quad \text{and} \quad \mu + \rho \le my + ry, \tag{7.6}$$

where entries ρ_{ij} may be negative. Define the splitting $\rho^+ := (|\rho| + \rho)/2$ and $\rho^- := (|\rho| - \rho)/2$, so that $\rho = \rho^+ - \rho^-$ with $0 \le \rho^-, \rho^+ \in \mathbb{R}^{m\times n}$ and $\rho^+ \circ \rho^- = 0$. Then there exists $\sigma \in \mathbb{R}^{m\times n}$ with $|\sigma| \le \rho^-$ and

$$\mu - E \circ \rho^+ - \sigma \in \langle my, ry\rangle \quad \text{for all } E \in \mathcal{E}. \tag{7.7}$$

Proof. Define $\sigma \in \mathbb{R}^{m\times n}$ entrywise by

$$\sigma_{ij} := \begin{cases} \mu_{ij} - (my + ry)_{ij} & \text{if } \rho^-_{ij} > 0 \text{ and } \mu_{ij} > (my + ry)_{ij}, \\ 0 & \text{if } \rho^-_{ij} = 0 \text{ or } (my - ry)_{ij} \le \mu_{ij} \le (my + ry)_{ij}, \\ \mu_{ij} - (my - ry)_{ij} & \text{if } \rho^-_{ij} > 0 \text{ and } \mu_{ij} < (my - ry)_{ij}. \end{cases} \tag{7.8}$$

Let $i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}$ and $s \in \mathbb{R}$ with $|s| \le 1$ be fixed but arbitrary. Then (7.6) and $\rho^+_{ij}\rho^-_{ij} = 0 = \rho^+_{ij}\sigma_{ij}$ imply

$$\mu_{ij} + s \cdot \rho^+_{ij} - \sigma_{ij} := \begin{cases} s \cdot \rho^+_{ij} + (my + ry)_{ij} = (my + ry)_{ij} \\ \mu_{ij} + s \cdot \rho^+_{ij} = \mu_{ij} + s \cdot \rho_{ij} \\ s \cdot \rho^+_{ij} + (my - ry)_{ij} = (my - ry)_{ij} \end{cases} \tag{7.9}$$

for the three cases as in (7.8). With (7.6) always $\mu_{ij} + s \cdot \rho_{ij}^{+} - \sigma_{ij} \in \langle my_{ij}, ry_{ij} \rangle$, and therefore (7.7). Moreover (7.6) yields

$$|\sigma_{ij}| = \begin{cases} \mu_{ij} - (my + ry)_{ij} \le -\rho_{ij} = \rho_{ij}^{-} & \text{if } \rho_{ij}^{-} > 0 \text{ and } \mu_{ij} > (my + ry)_{ij}, \\ 0 & \text{if } \rho_{ij}^{-} = 0 \text{ or } (my - ry)_{ij} \le \mu_{ij} \le (my + ry)_{ij}, \\ (my - ry)_{ij} - \mu_{ij} \le -\rho_{ij} = \rho_{ij}^{-} & \text{if } \rho_{ij}^{-} > 0 \text{ and } \mu_{ij} < (my - ry)_{ij}. \end{cases}$$

This completes the proof. □

Let $\mathbf{A} \in \mathbb{IR}^{n \times n}$ and $\mathbf{b} \in \mathbb{IR}^{n}$ be given with

$$\mathbf{A} = [A_{\inf}, A_{\sup}] = \langle mA, rA \rangle \quad \text{and} \quad \mathbf{b} = [b_{\inf}, b_{\sup}] = \langle mb, rb \rangle \tag{7.10}$$

for suitable $A_{\inf}, A_{\sup}, mA, rA \in \mathbb{R}^{n \times n}$ and $b_{\inf}, b_{\sup}, mb, rb \in \mathbb{R}^{n}$, where $rA, rb \ge 0$. Given $\widetilde{x} \in \mathbb{R}^{n}$ define

$$\mathbf{y} = \langle my, ry \rangle := \left[b_{\inf} - mA \cdot \widetilde{x} - rA \cdot |\widetilde{x}|, b_{\sup} - mA \cdot \widetilde{x} + rA \cdot |\widetilde{x}| \right] \in \mathbb{IR}^{n}. \tag{7.11}$$

Let fixed but arbitrary $y \in \mathbf{y}$ be given, and define a signature matrix T such that $T\widetilde{x} = |\widetilde{x}|$. Using (7.5) implies $y = my + S \cdot ry$ for some $S \in \mathcal{S}$, so that

$$y = my + S \cdot ry = mb - mA \cdot \widetilde{x} + S \cdot (rb + rA \cdot |\widetilde{x}|) = mb + S \cdot rb - (mA - S \cdot rA \cdot T)\widetilde{x}.$$

Hence $mb + S \cdot rb \in \mathbf{b}$ and $mA - S \cdot rA \cdot T \in \mathbf{A}$ yield

$$my - ry \le y \le my + ry \Leftrightarrow \exists A \in \mathbf{A}\, \exists b \in \mathbf{b} : y = b - A\widetilde{x}, \tag{7.12}$$

where the reverse implication follows by the definition (7.11). This is the basis to compute an inner inclusion by the following code.

Algorithm 7.2. Rigorous inner bounds `yinf, ysup` $\in \mathbb{F}^{n}$ of the solution of the linear systems $Ax = b$ the data of which are afflicted with tolerances.

```
 1  if tol && ( ( nargin==2 ) || ( ~Illco ) )      % compute inner inclusion
 2    mA = -(-inf_(A)-sup(A))/2; rA = -(inf_(A)-mA); % inner inclusion of A
 3    resinf = inf_(b) + mA*(-xs) + rA*(-abs(xs)); % lower inner bound
 4    setround(-1)                                 % rounding to downwards
 5    ressup = sup(b) + mA*(-xs) + rA*abs(xs);     % upper inner bound
 6    mu = resinf + 0.5*(ressup-resinf);           % inner midpoint
 7    rho = mu - resinf;                           % inner radius, maybe negative
 8    csup = R*mu + abs(R)*rho;                    % upper inner bound correction
 9    setround(1)                                  % rounding to upwards
10    cinf = R*mu + abs(R)*(-rho);                 % lower inner bound correction
11    e = mag(speye(n)-C)*err;                     % inner correction
12    yinf = xs + cinf + e;                        % inner lower bound
13    ysup = -(e - csup - xs);                     % inner upper bound
14   end
```

Theorem 7.3. *Let* $\mathtt{A} \in \mathbb{IF}^{n \times n}$ *and* $\mathtt{b} \in \mathbb{IF}^{n}$ *be given. Suppose Algorithm 4.2 (`LssErrBnd`) arrives at line* 30 *after the call* `x = LssErrBnd(A, b, 0)`. *Then, as by Theorem 4.3,* $\mathtt{x} \in \mathbb{IF}^{n}$ *computed in line* 29 *is an inclusion of the solution set* $\Sigma(\mathtt{A}, \mathtt{b})$, *and all* $A \in \mathtt{A}$ *are non-singular. Suppose the code in Algorithm 7.2 is inserted and executed between lines* 29 *and* 30 *in Algorithm 4.2 (`LssErrBnd`). Then for all* $i \in \{1, \ldots, n\}$ *there exist* $A_1, A_2 \in \mathtt{A}$ *and* $b_1, b_2 \in \mathtt{b}$ *with*

$$(A_1^{-1} b_1)_i \le \underset{i}{\mathtt{yinf}} \quad \text{and} \quad \underset{i}{\mathtt{ysup}} \le (A_2^{-1} b_2)_i. \tag{7.13}$$

The additional computational effort to compute inner inclusions is $17n^2 + \mathcal{O}(n)$ *floating-point operations.*

Remark 1. Note that the code in Algorithm 7.2 is only executed if the matrix and/or the right hand side contain data with tolerances, and not the option `Illco` $= 1$ for extremely ill-conditioned matrices was chosen. This is because otherwise the solution set has empty interior, or the sensitivity does not permit the computation of an inner inclusion, respectively.

Remark 2. Also note that (7.13) is valid even if $\mathtt{yinf}_i > \mathtt{ysup}_i$. In that case the inner inclusion is empty for the i-th component of $\Sigma(\mathtt{A}, \mathtt{b})$. However, at least some information is provided on the lower and upper bound of any $\hat{x}_i$ with $\hat{x} \in \Sigma(\mathtt{A}, \mathtt{b})$. An example is given in (7.30).

Proof of Theorem 7.3. We refer to the line numbering in Algorithm 7.2, and we use the midpoint-radius representation $\mathtt{A} =: \mathbf{A} = \langle mA, rA \rangle \in \mathbb{IR}^{n \times n}$ and $\mathtt{b} =: \mathbf{b} = \langle mb, rb \rangle \in \mathbb{IR}^{n}$ as in (7.10). Note that `inf_(x)`[3] and `sup(x)` for a (real) interval

[3] For syntactical reasons, `inf(x)` cannot be used in Matlab/INTLAB because `inf` is reserved for the constant ∞.

quantity x in INTLAB is only accessing the lower and upper bound, respectively, so that $\texttt{inf(A)} = A_{\inf}$ and so forth without rounding error. We split the computations in line 2 into

```
mA1 = (-inf_(A)-sup(A))/2; mA = -mA1;
rA1 = inf_(A)-mA; rA = -rA1;
```

The rounding mode in line 2 is still to upwards as set in line 18 in Algorithm 4.2 (LssErrBnd), so that

$$\texttt{mA1} \geq (-A_{\inf} - A_{\sup})/2 \quad \text{and} \quad \texttt{rA1} \geq A_{\inf} - \texttt{mA}.$$

Negation is error-free, so that

$$\texttt{mA} \leq (A_{\inf} + A_{\sup})/2 = mA \quad \text{and} \quad \texttt{rA} \leq \texttt{mA} - A_{\inf} \leq mA - A_{\inf} = rA. \tag{7.14}$$

Hence

$$mA - rA = A_{\inf} \leq \texttt{mA} - \texttt{rA} \quad \text{and} \quad \texttt{mA} + \texttt{rA} \leq mA + rA. \tag{7.15}$$

Therefore (7.6) is satisfied for $\mathbf{y} := \langle mA, rA\rangle$, $\mu := \texttt{mA}$ and $\rho := \texttt{rA}$. Splitting $\texttt{rA} = \texttt{rA}^+ - \texttt{rA}^-$ and using Lemma 7.1 there exists $\Sigma \in \mathbb{R}^{n\times n}$ with $|\Sigma| \leq \texttt{rA}^-$ and

$$\texttt{mA} - \texttt{rA}^+ \cdot S - \Sigma \in \langle mA, rA\rangle \quad \text{for all } S \in \mathcal{S}. \tag{7.16}$$

Setting $\widetilde{x} := \texttt{xs}$ and using $S \in \mathcal{S}$ such that $S\widetilde{x} = -|\widetilde{x}|$, rounding to upwards and line 3 in Algorithm 7.2 imply

$$\texttt{resinf} \geq b_{\inf} + \texttt{mA} \cdot (-\widetilde{x}) + \texttt{rA} \cdot (-|\widetilde{x}|) = b_{\inf} - \texttt{mA} \cdot \widetilde{x} + \texttt{rA}^+ S\widetilde{x} + \texttt{rA}^- |\widetilde{x}|. \tag{7.17}$$

By (7.16) and $\Sigma \leq |\Sigma| \leq \texttt{rA}^-$ there exist $A_1 \in \mathbf{A}$ and $b_1 \in \mathbf{b}$ with

$$\texttt{resinf} \geq b_{\inf} - (\texttt{mA} - \texttt{rA}^+ S - \Sigma) \cdot \widetilde{x} = b_1 - A_1\widetilde{x}. \tag{7.18}$$

Similarly, using rounding to downwards, we conclude for $S \in \mathcal{S}$ with $S\widetilde{x} = |\widetilde{x}|$ and line 5 in Algorithm 7.2 that

$$\texttt{ressup} \leq b_{\sup} + \texttt{mA} \cdot (-\widetilde{x}) + \texttt{rA} \cdot |\widetilde{x}| = b_{\sup} - \texttt{mA} \cdot \widetilde{x} + \texttt{rA}^+ S\widetilde{x} - \texttt{rA}^- |\widetilde{x}|, \tag{7.19}$$

so that using $-\texttt{rA}^- \leq -|\Sigma| \leq \Sigma$ there exist $A_2 \in \mathbf{A}$ and $b_2 \in \mathbf{b}$ with

$$\texttt{ressup} \leq b_{\sup} - (\texttt{mA} - \texttt{rA}^+ S - \Sigma) \cdot \widetilde{x} = b_2 - A_2\widetilde{x}. \tag{7.20}$$

Lines 6 and 7 and rounding to downwards imply

$$\texttt{mu} \leq \texttt{resinf} + (\texttt{ressup} - \texttt{resinf})/2 = (\texttt{resinf} + \texttt{ressup})/2$$
$$\texttt{rho} \leq \texttt{mu} - \texttt{resinf},$$

so that (7.18) and (7.20) yield

$$b_1 - A_1\widetilde{x} \leq \texttt{resinf} \leq \texttt{mu} - \texttt{rho} \quad \text{and} \quad \texttt{mu} + \texttt{rho} \leq 2\texttt{mu} - \texttt{resinf} \leq \texttt{ressup} \leq b_2 - A_2\widetilde{x}. \tag{7.21}$$

Hence for $\mathbf{y} = \langle my, ry\rangle$ defined in (7.11) we use (7.12) to conclude

$$my - ry \leq b_1 - A_1\widetilde{x} \leq \texttt{mu} - \texttt{rho} \quad \text{and} \quad \texttt{mu} + \texttt{rho} \leq b_2 - A_2\widetilde{x} \leq my + ry. \tag{7.22}$$

Abbreviating $\mu := \texttt{mu}$ and $\rho := \texttt{rho}$, Lemma 7.1 implies for the splitting $\rho = \rho^+ - \rho^-$ that there exist some $\sigma \in \mathbb{R}^n$ with $|\sigma| \leq \rho^-$ and $\mu - S \cdot \rho^+ - \sigma \in \langle my, ry\rangle$ for all $S \in \mathcal{S}$, and (7.12) gives

$$\forall S \in \mathcal{S}\ \exists A \in \mathbf{A}\ \exists b \in \mathbf{b} : b - A\widetilde{x} = \mu - S \cdot \rho^+ - \sigma. \tag{7.23}$$

For the remainder of the proof let $i \in \{1, \ldots, n\}$ be fixed but arbitrary. For $R \in \mathbb{R}^{n\times n}$ define signature matrices $|S_1| = |S_2| = I$ such that $(RS_1)_{ik} \geq 0$ and $(RS_2)_{ik} \leq 0$ for all $k \in \{1, \ldots, n\}$. By (7.23) there exist $A_1, A_2 \in \mathbf{A}$ and $b_1, b_2 \in \mathbf{b}$ with $b_\nu - A_\nu\widetilde{x} = \mu - S_\nu\rho^+ - \sigma$ for $\nu \in \{1, 2\}$. Then

$$\begin{aligned} \left[R(b_1 - A_1\widetilde{x})\right]_i &= \left[R\mu - |R|\rho^+ - R\sigma\right]_i \leq \left[R\mu - |R|\rho^+ + |R|\rho^-\right]_i = \left[R\mu - |R|\rho\right]_i \quad \text{and} \\ \left[R(b_2 - A_2\widetilde{x})\right]_i &= \left[R\mu + |R|\rho^+ - R\sigma\right]_i \geq \left[R\mu + |R|\rho^+ - |R|\rho^-\right]_i = \left[R\mu + |R|\rho\right]_i. \end{aligned} \tag{7.24}$$

For $R := \texttt{R}$ the computation of csup and cinf in lines 8 and 10 and the rounding modes in use imply

$$\texttt{R} \cdot \texttt{mu} - |\texttt{R}| \cdot \texttt{rho} \leq \texttt{cinf} \quad \text{and} \quad \texttt{csup} \leq \texttt{R} \cdot \texttt{mu} + |\texttt{R}| \cdot \texttt{rho},$$

so that by (7.24) and the definition of μ and ρ

$$\left[R(b_1 - A_1\widetilde{x})\right]_i \leq \texttt{cinf}_i \quad \text{and} \quad \texttt{csup}_i \leq \left[R(b_2 - A_2\widetilde{x})\right]_i. \tag{7.25}$$

For fixed but arbitrary $A \in \mathbf{A}$, $b \in \mathbf{b}$ define $C := RA$ and $c := R(A\widetilde{x} - b)$, so that, as in (2.1) and (2.2),

$$A^{-1}b = \widetilde{x} - C^{-1}c = \widetilde{x} - c - (I - C)C^{-1}c. \tag{7.26}$$

For the quantities `xs` and `err` computed before line 29 in Algorithm 4.2 (`LssErrBnd`) we know by Theorem 4.3 that $|A^{-1}b - \mathtt{xs}| \le \mathtt{err}$ for all $A \in \mathbf{A}$ and $b \in \mathbf{b}$. This is in particular true for $A_1, A_2 \in \mathbf{A}$ and $b_1, b_2 \in \mathbf{b}$ as in (7.25). Moreover, $RA \in \mathtt{C}$ for all $A \in \mathbf{A}$ for `C` as computed in line 12 in Algorithm 4.2, so that the computation of `e` in line 11 of Algorithm 7.2 in rounding to upwards and `mag` as defined in Table 4.1 imply $|I - RA \mathbin{\|} C^{-1}c| \le \mathtt{e}$. By (7.26) and $\mathtt{xs} = \widetilde{x}$ it follows that `yinf` computed in line 12 of Algorithm 7.2 satisfies

$$\left[A_1^{-1}b_1\right]_i = \widetilde{x}_i + \left[R(b_1 - A_1\widetilde{x})\right]_i - \left[(I - RA_1)(\widetilde{x} - A_1^{-1}b_1)\right]_i \le \mathtt{xs}_i + \underset{i}{\mathtt{cinf}} + \mathtt{e} \le \underset{i}{\mathtt{yinf}}. \tag{7.27}$$

Now `ysup` is computed by $\mathtt{z} = \mathtt{e} - \mathtt{csup} - \mathtt{xs}$ and $\mathtt{ysup} = -\mathtt{z}$ in rounding to upwards. Therefore $\mathtt{z} \ge \mathtt{e} - \mathtt{csup} - \mathtt{xs}$ and $\mathtt{ysup} \le \mathtt{xs} + \mathtt{csup} - \mathtt{e}$, so that similar to the previous conclusion we obtain

$$\left[A_2^{-1}b_2\right]_i = \widetilde{x}_i + \left[R(b_2 - A_2\widetilde{x})\right]_i - \left[(I - RA_2)(\widetilde{x} - A_2^{-1}b_2)\right]_i \ge \mathtt{xs}_i + \underset{i}{\mathtt{csup}} - \mathtt{e} \ge \underset{i}{\mathtt{ysup}}. \tag{7.28}$$

This completes the proof because i was chosen fixed but arbitrary in $\{1, \ldots, n\}$. □

Coming back to our toy example (7.1), Algorithm 4.2 (`LssErrBnd`) together with Algorithm 7.2 compute outer and inner bounds as follows, rounded to 2 decimals:

$$\begin{pmatrix}[-0.35, 0.18]\\ [-0.27, 0.00]\end{pmatrix} \subseteq \mathrm{hull}\big(\Sigma(\mathbf{A}, \mathbf{b})\big) \subseteq \begin{pmatrix}[-0.56, 0.40]\\ [-0.57, 0.30]\end{pmatrix}. \tag{7.29}$$

The situation is depicted in Fig. 7.1. As can be seen there is some distance between the inner and outer inclusion. In this case the latter is in fact not far from the optimal inclusion:

$$\mathrm{hull}\big(\Sigma(\mathbf{A}, \mathbf{b})\big) = \begin{pmatrix}[-0.48, 0.30]\\ [-0.55, 0.19]\end{pmatrix} \quad \text{to 2 decimal places.}$$

In this toy example it is not difficult to compute the (optimal) interval hull of the solution set. Note that in general this is an NP-hard problem, and it is NP-complete to decide that an interval matrix contains a singular matrix [26].

For wider input data, some or all of the entries of the inner inclusion may become empty. If, for instance, $\mathbf{A}_{11}$ in (7.1) is changed from $[2, 2]$ to $[1, 1.75]$, then the second entry of the inner inclusion becomes empty (all data rounded to 2 decimal places):

$$\begin{pmatrix}[-0.15, -0.07]\\ - - -\end{pmatrix} \subseteq \mathrm{hull}\big(\Sigma(\mathbf{A}, \mathbf{b})\big) = \begin{pmatrix}[-0.93, 0.50]\\ [-0.84, 0.34]\end{pmatrix} \subseteq \begin{pmatrix}[-1.2, 0.93]\\ [-0.96, 0.65]\end{pmatrix}. \tag{7.30}$$

Although the inner inclusion for the second component of $\Sigma(\mathbf{A}, \mathbf{b})$ is empty because $\mathtt{yinf}_2 = -0.09$ and $\mathtt{ysup}_2 = -0.21$, at least the information

$$\inf\big(\mathrm{hull}\big(\Sigma(\mathbf{A}, \mathbf{b})\big)_2\big) \le -0.09 \quad \text{and} \quad -0.21 \le \sup\big(\mathrm{hull}\big(\Sigma(\mathbf{A}, \mathbf{b})\big)_2\big) \tag{7.31}$$

is obtained. Note that nevertheless there is not too much difference between the optimal inclusion and the bounds computed by Algorithm 4.2 (`LssErrBnd`). For narrow input intervals this is typical and can be explained as follows. Up to rounding errors, $[\mathtt{resinf}, \mathtt{ressup}]$ is equal to $\mathbf{y}$ as defined in (7.11), and also up to rounding errors $[\mathtt{cinf}, \mathtt{csup}]$ is equal to the hull of $\{\mathtt{R} \cdot r : r \in [\mathtt{resinf}, \mathtt{ressup}]\}$. Hence the difference of the inner and outer inclusion is determined by the size of `e`, which estimates $|(I - C)C^{-1}c|$ as by (7.26). This in turn is the product of $I - R \cdot \mathtt{A}$ and the maximum difference of $A^{-1}b$ and $\widetilde{x}$ for $A \in \mathtt{A}$ and $b \in \mathtt{b}$. If the tolerances of the matrix entries are not too large, then this is the product of two small quantities. In particular it almost vanishes if the input matrix `A` has no tolerances at all.

This implies that the inner and outer inclusion are almost identical if there are only tolerances in the right hand side. As an example, we generate random matrices of different dimensions and condition number, and some random right hand side by b with entries from a normal distribution with mean zero and standard deviation one. Then each entry b_i is afflicted with a tolerance $b_i \pm r_i$, where r_i is taken from a uniform distribution in $[0, 1]$. Note that this implies that the radius of the solution set is roughly of the order of the condition number. The ratio of the median of the radii of the inner and outer inclusion for different dimensions and condition numbers is displayed in Fig. 7.2. A ratio close to 1 means that the inner and outer inclusions are almost identical. As can be seen the ratio becomes less than one only for large condition numbers, reflecting the increasing influence of accumulated rounding errors.

As has been mentioned, the previous approaches in [35–37] basically compute inner inclusions only if all entries in the matrix $\mathbf{A}$ and the right hand side $\mathbf{b}$ are intervals with non-empty interior. Theorem 7.3 is applicable if only one entry in the matrix or the right hand side is a proper interval. A typical example is as follows. We generate a random matrix $A \in \mathbb{R}^{1000\times 1000}$

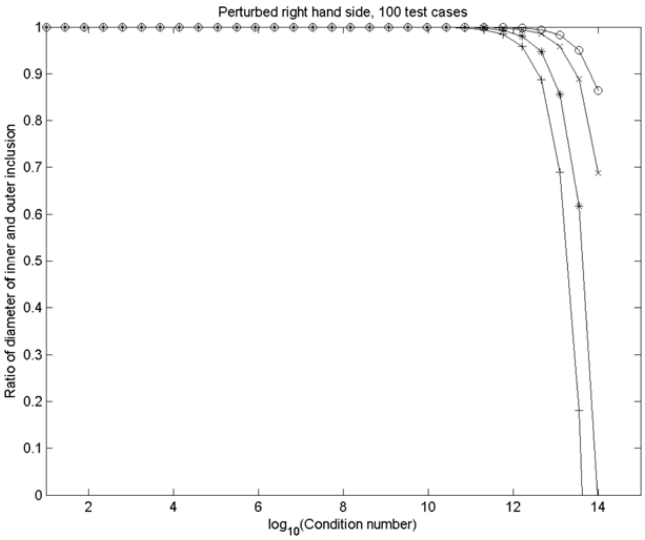

Fig. 7.2. Median ratio of radii of inner and outer inclusion for perturbations in the right hand side, dimensions $n = 100$ (o), $n = 200$ ($\times$), $n = 500$ ($*$) and $n = 1000$ ($+$).

with random right hand side and replace some random entry like $A_{529,386}$ by $[-|A_{529,386}|, |A_{529,386}|]$. Then a typical outer and inner inclusion of any solution component may be as follows:

$$[0.5158, 0.5564] \subseteq \text{hull}\big(\Sigma(\mathbf{A}, \mathbf{b})\big)_{817} \subseteq [0.5143, 0.5579].$$

In many test examples the result were similar or better, i.e. less difference between the inner and outer inclusion. For smaller uncertainties like replacing $A_{913,209}$ by $A_{913,209} \cdot (1 \pm 0.1)$ instead, there is almost no difference between the outer and inner inclusion. This allows, for example, to prove

$$\text{hull}\big(\Sigma(\mathbf{A}, \mathbf{b})\big)_{308} = [-0.9731, -0.9687]$$

up to four decimal digits. The results for the other components are completely similar.

8. Computational results

Following we report computational results. All algorithms are tested in Matlab version 7.11.0.584 (R2010b) on an Intel Core i7 CPU M640 with 2.8 GHz, INTLAB version 6 and Windows 7 operating system. We discuss the numerical behavior for the following algorithms:

	method
I	Algorithm 4.2 (`LssErrBnd`) with the improvement described in Subsection 3.3
II	Algorithm 4.16 (`LssErrBndNear`) from Part I of this paper
III	`verifylss` in INTLAB (see Subsection 5.1)
IV	Nguyen's method (see Subsection 5.4)
V	Ogita's method (see Subsection 5.3)
VI	Theorem 5.2

(8.1)

To allow a fair comparison, all algorithms are written in Matlab/INTLAB, and all algorithms use the same extra-precise dot product accumulation. For the vector residuals all algorithms use `Dot_` as provided in INTLAB rather than `Dot2`. Mathematically `Dot_` is identical to Algorithm 4.1 (`Dot2`) but suffers less from interpretation overhead due to a tricky vectorization of operations. `Dot_` is only suited for vector residuals; for the matrix residual in `LssErrBnd` for extremely ill-conditioned problems, `Dot2` is used.

We use the source code of `LssErrBnd` and `LssErrBndNear` as presented in Parts II and I of this paper, respectively; only `LssErrBnd` is improved as described in Section 3.3. This gains some computing time for not too ill-conditioned problems. Nguyen and Ogita kindly provided the Matlab/INTLAB source code of their routines `certifylss_relaxed` [33] and `vlssr` [30], respectively. The improved version of Neumaier's algorithm based on Hansen's method in Theorem 5.2 using (5.12) and (5.13) we implemented ourselves. Algorithm 3.1 (`LssErrBndDirRdg`) was presented to display an algorithm using only rounding to nearest and avoiding interval operations. Since the results are identical to those of `LssErrBnd`, they need not to be discussed separately.

To our knowledge the only other Matlab codes for computing rigorous bounds for the solution of linear systems are by Hargreaves [19] and Rohn [38]. The first author admits that `verifylss` is superior to his routine; the second author provides a library for computing rigorous error bounds for a number of basic problems in linear algebra. However, for square linear systems he uses `verifylss`.

Table 8.1
Median relative error of the bounds computed by Algorithm 4.2 (`LssErrBnd`) [I], Algorithm 4.16 (`LssErrBndNear`) from Part I of this paper [II], `verifylss` in INTLAB [III], Nguyen's method [IV], Ogita's method [V], and Theorem 5.2 [VI] for the matrices from Table 5.1 in Part I and right hand side `b = A * randn(n, 1)`.

Matrix	n	cond(A)	I	II	III	IV	V	VI
Pascal	17	$2.2 \cdot 10^{16}$	$1.5 \cdot 10^{-16}$	$2.8 \cdot 10^{-16}$	$1.5 \cdot 10^{-16}$	$1.5 \cdot 10^{-16}$	$7.0 \cdot 10^{-16}$	$1.5 \cdot 10^{-16}$
	18	$2.5 \cdot 10^{17}$	$2.0 \cdot 10^{-16}$	–	$2.0 \cdot 10^{-16}$	$2.0 \cdot 10^{-16}$	–	$2.0 \cdot 10^{-16}$
Hilbert	11	$2.5 \cdot 10^{14}$	$1.8 \cdot 10^{-16}$	$1.8 \cdot 10^{-16}$	$1.6 \cdot 10^{-16}$	$1.6 \cdot 10^{-16}$	$4.0 \cdot 10^{-16}$	$1.8 \cdot 10^{-16}$
	12	$8.8 \cdot 10^{15}$	$1.0 \cdot 10^{-14}$	–	$1.8 \cdot 10^{-12}$	$4.5 \cdot 10^{-14}$	–	$4.9 \cdot 10^{-15}$
ScHilbert	11	$3.0 \cdot 10^{14}$	$1.8 \cdot 10^{-16}$	$1.8 \cdot 10^{-16}$	$1.8 \cdot 10^{-16}$	$1.8 \cdot 10^{-16}$	$1.1 \cdot 10^{-15}$	$1.8 \cdot 10^{-16}$
	12	$7.8 \cdot 10^{15}$	$3.2 \cdot 10^{-16}$	–	$2.0 \cdot 10^{-16}$	$3.6 \cdot 10^{-14}$	–	$2.6 \cdot 10^{-16}$
InvHilbert	11	$2.9 \cdot 10^{14}$	$1.4 \cdot 10^{-16}$	$1.4 \cdot 10^{-16}$	$1.4 \cdot 10^{-16}$	$1.4 \cdot 10^{-16}$	$6.3 \cdot 10^{-16}$	$1.4 \cdot 10^{-16}$
	12	$1.0 \cdot 10^{16}$	$2.0 \cdot 10^{-16}$	–	$2.0 \cdot 10^{-16}$	$3.5 \cdot 10^{-13}$	$1.7 \cdot 10^{-13}$	$2.0 \cdot 10^{-16}$
Boothroyd	11	$1.9 \cdot 10^{15}$	$1.5 \cdot 10^{-16}$	$1.9 \cdot 10^{-16}$	$1.5 \cdot 10^{-16}$	$1.5 \cdot 10^{-16}$	$3.5 \cdot 10^{-15}$	$1.5 \cdot 10^{-16}$
	12	$8.5 \cdot 10^{16}$	$2.7 \cdot 10^{-15}$	–	$1.7 \cdot 10^{-16}$	$8.9 \cdot 10^{-14}$	–	$2.2 \cdot 10^{-15}$
Vander	12	$3.0 \cdot 10^{14}$	$1.5 \cdot 10^{-16}$	$1.5 \cdot 10^{-16}$	$1.5 \cdot 10^{-16}$	$1.5 \cdot 10^{-16}$	$2.0 \cdot 10^{-16}$	$1.5 \cdot 10^{-16}$
	13	$1.3 \cdot 10^{16}$	$1.9 \cdot 10^{-16}$	$3.1 \cdot 10^{-16}$	$1.9 \cdot 10^{-16}$	$1.6 \cdot 10^{-16}$	$2.1 \cdot 10^{-16}$	$1.9 \cdot 10^{-16}$

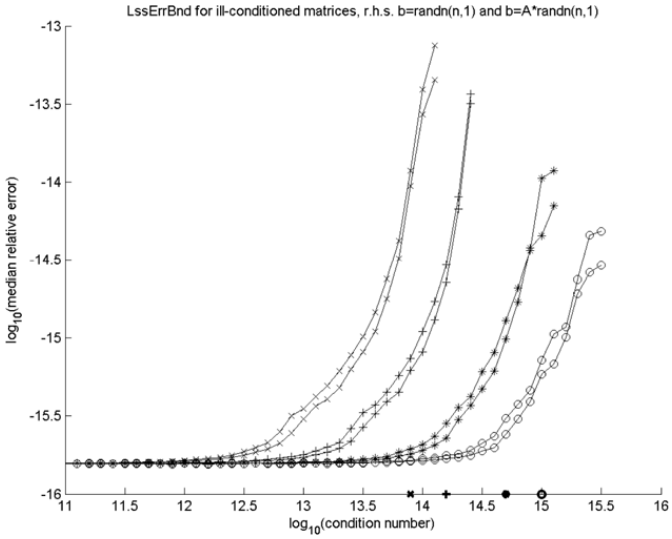

Fig. 8.1. Results of Algorithm 4.2 (`LssErrBnd`) for ill-conditioned random matrices of dimension $n = 100$ (o), $n = 200$ ($*$), $n = 500$ ($+$) and $n = 1000$ ($\times$).

To begin with, we display in Table 8.1 the results for the "usual suspects" of ill-conditioned matrices as listed in Table 5.1 in Part I.[4] Almost always in the following the results for the right hand side `b = randn(n, 1)` and `b = A * randn(n, 1)` are practically identical, thus we mostly display only one of them. Recall that the Matlab function `rand` generates pseudo-random values drawn from a uniform distribution on the unit interval, whereas `randn` produces pseudo-random values drawn from a normal distribution with mean zero and standard deviation one. In Table 8.1 we use `b = A * randn(n, 1)`. The largest dimensions are displayed for which at least one of Algorithms I–VI succeeds. Here and in the following always the median of the relative error of the inclusions is displayed.

As can be seen, `LssErrBnd` [I] and Theorem 5.2 [VI] are almost always maximally accurate, `verifylss` [III] and Nguyen's method [IV] are sometimes slightly less accurate. Those four methods have all the same scope of applicability (which is clear for methods [I] and [VI] because they use the same Perron vector) of about $\text{cond}(A) \lesssim \mathbf{u}^{-1}$. `LssErrBndNear` [II] and Ogita's method [V] fail for some ill-conditioned test matrices. For `LssErrBndNear` this is because all error estimations are performed in rounding to nearest, whereas Ogita's method could be improved by using a Perron vector to scale $I - RA$. We did not do this but used the original algorithm provided by the author.

For condition numbers up to $\mathbf{u}^{-1}$, matrices are generated by `randsvd(n,cnd)` from the Matlab matrix gallery. Here some random orthogonal transformation is applied from the left and right to a diagonal matrix with specified singular values. In Fig. 8.1 the median of the relative errors of the inclusion computed by Algorithm 4.2 (`LssErrBnd`) [I] is displayed for 100 test cases for each dimension, for different condition numbers, and for right hand sides `b = randn(n, 1)` and `b = A * randn(n, 1)`. Only results for condition numbers larger than 10^{11} are displayed; for smaller condition numbers the

[4] For a discussion of those "well-known suspects" of ill-conditioned matrices A and alternative ways to solve $Ax = b$ see the beginning of Section 5 in Part I.

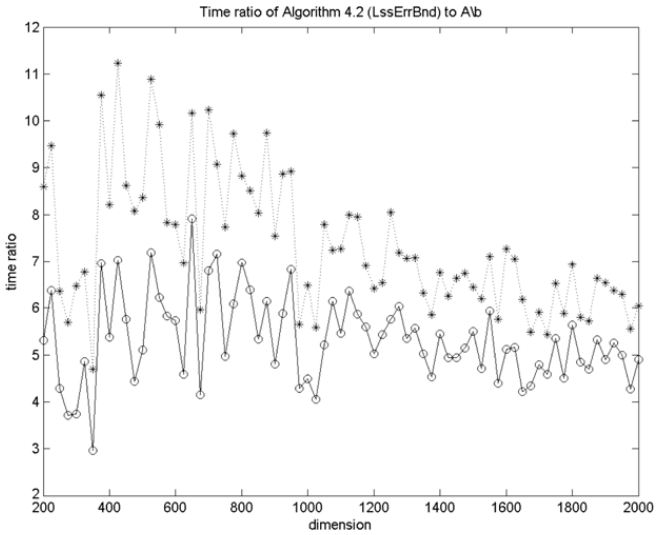

Fig. 8.2. Time ratio between Algorithm 4.2 (`LssErrBnd`) and the standard Matlab call `A \ b`, the former with extra-precise evaluation of dot products (dotted line, $(*)$) and with dot products in working precision (solid line, (o)).

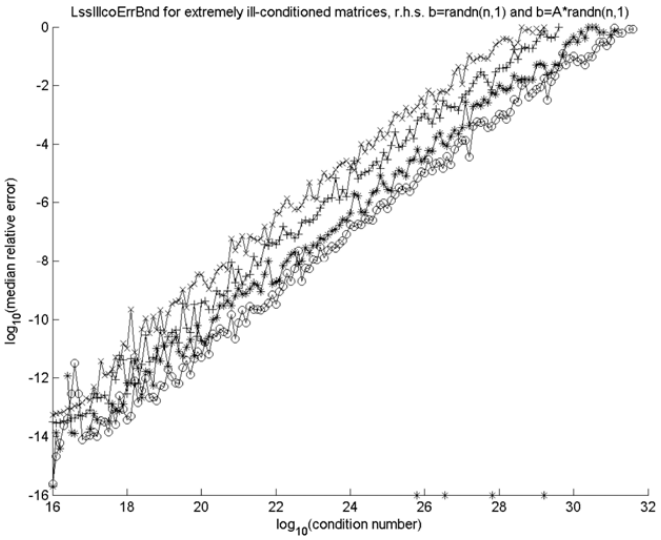

Fig. 8.3. Results of Algorithm 4.2 (`LssErrBnd`) for extremely ill-conditioned random matrices of dimension $n = 100$ (o), $n = 200$ $(*)$, $n = 500$ $(+)$ and $n = 1000$ $(\times)$.

results are always of maximum accuracy. Compared to Fig. 5.4 in Part I for `LssErrBndNear` the curves are shifted to the right and show the advantage of using directed rounding.

For $n = 100, n = 200, n = 500$ and $n = 1000$, `LssErrBnd` did not fail in all test cases for $\mathrm{cond}(A) \leq 1.0 \cdot 10^{15}$, $1.0 \cdot 10^{14}$, $1.6 \cdot 10^{14}$ and $7.9 \cdot 10^{13}$, which is better than $\mathbf{u}^{-1}/n$. Recall that the applicability of Algorithm 4.16 (`LssErrBndNear`) in Part I was roughly limited to condition numbers up to $\mathbf{u}^{-1}/n^2$.

The computing time of `LssErrBnd` [I] is similar to `LssErrBndNear` [II] (see subsequent tables). In Fig. 8.2 the time ratio between Algorithm 4.2 (`LssErrBnd`) and the Matlab command $A \setminus b$ is displayed for 100 random matrices of dimensions from 100 to 2000. Here apples and oranges are compared for two reasons. First, $A \setminus b$ computes an approximation compared to rigorous error bounds by `LssErrBnd`, and second $A \setminus b$ uses plain Gaussian elimination without residual iteration in contrast to `LssErrBnd`. To make it a little more fair, Fig. 8.2 shows the ratio for `LssErrBnd` as specified in Algorithm 4.2 with extra-precise residual correction in the upper curve $(*)$, and with only one residual correction in working precision (ensuring backward stability [39]) in the lower curve (o). The latter inclusions are less accurate, as is $A \setminus b$.

With extra-precise residual corrections, the factor in computing time between `LssErrBnd` and $A \setminus b$ is about 6–7, whereas without the factor it improves to about 5. Note that by Theorem 4.3 the theoretical ratio is 9 for `LssErrBnd` as stated, and 6 with the improvement discussed in Section 3.3.

In Fig. 8.3 results of Algorithm 4.2 (`LssErrBnd`) for extremely ill-conditioned matrices are shown. We use the methods as described in Part I to generate extremely ill-conditioned matrices. Comparing Fig. 5.6 in Part I and Fig. 8.3 in this part, an improvement of applicability of `LssErrBnd` over Algorithm 4.16 (`LssIllcoErrBndNear`) in Part I, similar to Table 8.1 for small dimension, is observed. The latter succeeds to compute rigorous inclusions for condition numbers up to about $\mathbf{u}^{-2}/n^2$,

Table 8.2
Results of Algorithm 4.2 (`LssErrBnd`) [I], Algorithm 4.16 (`LssErrBndNear`) from Part I of this paper [II], `verifylss` in INTLAB [III], Nguyen's method [IV], Ogita's method [V], and Theorem 5.2 [VI], dimension $n = 100$, r.h.s. `randn(n,1)`, $K = 100$ test cases.

cond(A)	Median (median rel. error)						Percentage of success					
	I	II/I	III/I	IV/I	V/I	VI/I	I	II	III	IV	V	VI
10^1	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	12.82	1.00	100	100	100	100	100	100
10^4	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	1.00	1.00	100	100	100	100	100	100
10^7	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	1.00	1.00	100	100	100	100	100	100
10^{10}	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	18.06	1.00	100	100	100	100	100	100
10^{12}	$1.6 \cdot 10^{-16}$	1.01	1.00	1.00	1.64	1.00	100	100	100	100	100	100
10^{13}	$1.6 \cdot 10^{-16}$	1.09	1.00	1.00	15.26	1.00	100	100	100	100	100	100
10^{14}	$1.6 \cdot 10^{-16}$	92.38	1.00	0.96	20.24	1.00	100	33	100	100	100	100
$5 \cdot 10^{14}$	$2.4 \cdot 10^{-16}$	–	0.99	0.69	13.24	1.05	100	0	100	100	92	100
$6 \cdot 10^{14}$	$3.0 \cdot 10^{-16}$	–	1.00	0.57	12.75	1.09	100	0	100	100	84	100
$7 \cdot 10^{14}$	$3.6 \cdot 10^{-16}$	–	1.00	1.00	18.08	1.15	100	0	100	100	69	100
$8 \cdot 10^{14}$	$4.9 \cdot 10^{-16}$	–	1.00	$9 \cdot 10^{16}$	34.33	1.23	100	0	100	100	34	100
$9 \cdot 10^{14}$	$7.0 \cdot 10^{-16}$	–	0.99	$9 \cdot 10^{16}$	19.71	1.38	98	0	98	98	26	98
10^{15}	$1.1 \cdot 10^{-15}$	–	0.98	$9 \cdot 10^{16}$	45.47	1.42	96	0	96	94	21	96
$2 \cdot 10^{15}$	$1.1 \cdot 10^{-14}$	–	0.93	$9 \cdot 10^{16}$	–	1.76	30	0	30	23	0	30
$3 \cdot 10^{15}$	$1.4 \cdot 10^{-14}$	–	0.83	$9 \cdot 10^{16}$	–	1.98	4	0	4	2	0	4
$4 \cdot 10^{15}$	$2.1 \cdot 10^{-15}$	–	0.93	$9 \cdot 10^{16}$	–	1.98	2	0	2	2	0	2

Table 8.3
Results of Algorithm 4.2 (`LssErrBnd`) [I], Algorithm 4.16 (`LssErrBndNear`) from Part I of this paper [II], `verifylss` in INTLAB [III], Nguyen's method [IV], Ogita's method [V], and Theorem 5.2 [VI], dimension $n = 100$, r.h.s. `A * randn(n, 1)`, $K = 100$ test cases.

cond(A)	Median (median rel. error)						Percentage of success					
	I	II/I	III/I	IV/I	V/I	VI/I	I	II	III	IV	V	VI
10^1	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	16.43	1.00	100	100	100	100	100	100
10^4	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	1.00	1.00	100	100	100	100	100	100
10^7	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	1.00	1.00	100	100	100	100	100	100
10^{10}	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	26.34	1.00	100	100	100	100	100	100
10^{12}	$1.6 \cdot 10^{-16}$	1.01	1.00	1.00	5.11	1.00	100	100	100	100	100	100
10^{13}	$1.6 \cdot 10^{-16}$	1.19	1.00	0.99	2.87	1.00	100	100	100	100	100	100
10^{14}	$1.7 \cdot 10^{-16}$	110	1.03	0.94	4.13	1.00	100	30	100	100	100	100
$5 \cdot 10^{14}$	$2.9 \cdot 10^{-16}$	–	1.25	0.60	9.91	1.06	100	0	100	100	96	100
$6 \cdot 10^{14}$	$3.9 \cdot 10^{-16}$	–	1.24	3.04	14.85	1.11	100	0	100	100	81	100
$7 \cdot 10^{14}$	$4.2 \cdot 10^{-16}$	–	1.30	0.42	12.34	1.14	100	0	100	100	65	100
$8 \cdot 10^{14}$	$5.6 \cdot 10^{-16}$	–	1.34	$1.8 \cdot 10^{6}$	20.01	1.23	100	0	100	100	37	100
$9 \cdot 10^{14}$	$8.2 \cdot 10^{-16}$	–	1.34	$2.3 \cdot 10^{13}$	16.13	1.31	100	0	100	99	31	100
10^{15}	$1.1 \cdot 10^{-15}$	–	1.34	$2.5 \cdot 10^{14}$	22.09	1.44	96	0	96	96	17	96
$2 \cdot 10^{15}$	$6.9 \cdot 10^{-15}$	–	1.33	$1.1 \cdot 10^{15}$	–	1.85	37	0	37	28	0	37
$3 \cdot 10^{15}$	$3.4 \cdot 10^{-15}$	–	1.42	$2.4 \cdot 10^{14}$	–	2.01	6	0	6	5	0	6
$4 \cdot 10^{15}$	$1.1 \cdot 10^{-14}$	–	1.56	–	–	2.08	1	0	1	0	0	1

whereas `LssErrBnd` succeeds up to about $\mathbf{u}^{-2}/n$. Otherwise the graphs in Fig. 8.3 are similar to those in Fig. 5.6 in Part I but shifted to the right. Again it shows the improvement by using directed rounding.

In our examples, `LssErrBnd` succeeded for dimension $n = 100$, $n = 200$, $n = 500$ and $n = 1000$ in all test cases for $\text{cond}(A) \le 1.6 \cdot 10^{29}$, $6.6 \cdot 10^{27}$, $6.1 \cdot 10^{25}$ and $3.5 \cdot 10^{26}$, respectively (the value for $n = 500$ is smaller than for $n = 1000$ by the randomness of the examples).

Next we give detailed comparisons between the six methods listed in (8.1). First, for dimension $n = 100$ matrices with different condition numbers are generated 100 each. Then methods [I]–[VI] are applied to a linear system with right hand side `b = randn(n, 1)`. Denote by μ_k the median over the 100 test cases of the median relative errors of the result of Algorithm k. Then in Table 8.2 we display μ_1 for `LssErrBnd` [I] in column 2, and in columns 3–7 the ratio μ_k/μ_1 for $2 \le k \le 6$. Furthermore, in columns 8–13 the percentage of success out of the 100 test cases is displayed.

`LssErrBnd` [I] shows a slight decrease of accuracy with increasing condition number, but results guarantee at least 14 correct digits. For larger condition numbers, the inclusions of `verifylss` [III] are a little better (up to 17%), those of Theorem 5.2 [VI] a little weaker, and those by Ogita's method [V] provide about 1 decimal digit less accuracy. Nguyen's method [IV] is better for condition numbers around 10^{14} and significantly weaker for condition numbers up to $\mathbf{u}^{-1}$. Here the ratio $9 \cdot 10^{16}$ indicates that at least half of the inclusion components are intervals containing 0.

As expected, `LssErrBndNear` fails for condition numbers approaching $\mathbf{u}^{-1}/n$. The scope of applicability of methods [I] (and of course of [VI]) and [III] is the same, that of Nguyen's and Ogita's is a little less. Although in favor of `LssErrBnd`, we do not display computing times for small dimensions because the measurement error is too high.

For $n = 100$ we also display the results for right hand side `A*randn(n,1)` in Table 8.3 because the behavior of `verifylss` is a little different. Now for condition numbers close to $\mathbf{u}^{-1}$ the result is less accurate than `LssErrBnd`, the

Table 8.4
Results of Algorithm 4.2 (`LssErrBnd`) [I], Algorithm 4.16 (`LssErrBndNear`) from Part I of this paper [II], `verifylss` in INTLAB [III], Nguyen's method [IV], Ogita's method [V], and Theorem 5.2 [VI], dimension $n = 1000$, r.h.s. `randn(n,1)`, $K = 100$ test cases.

cond(A)	Median (median rel. error)						Relative comp. time				
	I	II/I	III/I	IV/I	V/I	VI/I	II/I	III/I	IV/I	V/I	VI/I
10^{1}	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	1.00	1.00	1.21	1.17	1.75	1.57	1.57
10^{4}	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	1.00	1.00	1.21	1.17	1.79	1.59	1.58
10^{7}	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	1.00	1.00	1.19	1.16	2.25	1.48	1.57
10^{10}	$1.6 \cdot 10^{-16}$	1.01	1.02	0.98	0.98	0.98	1.17	1.05	2.66	1.58	1.42
10^{12}	$1.6 \cdot 10^{-16}$	160	10.5	0.96	1.01	1.00	0.90*	0.88	2.75	1.55	1.22
$4 \cdot 10^{13}$	$1.4 \cdot 10^{-15}$	–	75.1	0.11	1.65	1.29	–	1.00	4.26	2.26	1.21
$5 \cdot 10^{13}$	$2.1 \cdot 10^{-15}$	–	77.4	0.08	2.31	1.50	–	1.04	4.51	2.44*	1.20
$6 \cdot 10^{13}$	$3.1 \cdot 10^{-15}$	–	76.6	0.05	5.17	1.80	–	1.18	4.92	2.50*	1.20
$7 \cdot 10^{13}$	$5.4 \cdot 10^{-15}$	–	73.8	0.05	7.73	2.21	–	1.51	4.96*	2.59*	1.19
$8 \cdot 10^{13}$	$1.0 \cdot 10^{-14}$	–	67.5	$3 \cdot 10^{8}$	34.5	2.53	–	1.54*	4.13*	2.48*	1.17*
$9 \cdot 10^{13}$	$2.5 \cdot 10^{-14}$	–	61.1	$2 \cdot 10^{14}$	–	2.59	–	1.62*	3.09*	–	1.17*
10^{14}	$5.1 \cdot 10^{-14}$	–	50.4	$3 \cdot 10^{14}$	–	2.45	–	1.58*	2.76*	–	1.17*
$2 \cdot 10^{14}$	–	–	–	–	–	–	–	–	–	–	–

* At the time ratio indicates that not for all 100 test cases inclusions were computed successfully.

Table 8.5
Results of Algorithm 4.2 (`LssErrBnd`) [I], Algorithm 4.16 (`LssErrBndNear`) from Part I of this paper [II], `verifylss` in INTLAB [III], Nguyen's method [IV], Ogita's method [V], and Theorem 5.2 [VI], dimension $n = 1000$, r.h.s. `randn(n,1)`, $K = 100$ test cases.

cond(A)	Median (median rel. error)						Percentage of success					
	I	II/I	III/I	IV/I	V/I	VI/I	I	II	III	IV	V	VI
10^{1}	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	1.00	1.00	100	100	100	100	100	100
10^{4}	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	1.00	1.00	100	100	100	100	100	100
10^{7}	$1.6 \cdot 10^{-16}$	1.00	1.00	1.00	1.00	1.00	100	100	100	100	100	100
10^{10}	$1.6 \cdot 10^{-16}$	1.01	1.02	0.98	0.98	0.98	100	100	100	100	100	100
10^{12}	$1.6 \cdot 10^{-16}$	160	10.50	0.96	1.01	1.00	100	91	100	100	100	100
$4 \cdot 10^{13}$	$1.4 \cdot 10^{-15}$	–	75.1	0.11	1.65	1.29	100	0	100	100	100	100
$5 \cdot 10^{13}$	$2.1 \cdot 10^{-15}$	–	77.4	0.08	2.31	1.50	100	0	100	100	97	100
$6 \cdot 10^{13}$	$3.1 \cdot 10^{-15}$	–	76.6	0.05	5.17	1.80	100	0	100	100	55	100
$7 \cdot 10^{13}$	$5.4 \cdot 10^{-15}$	–	73.8	0.05	7.73	2.21	100	0	100	99	9	100
$8 \cdot 10^{13}$	$1.0 \cdot 10^{-14}$	–	67.5	$3 \cdot 10^{8}$	34.5	2.53	87	0	87	80	5	87
$9 \cdot 10^{13}$	$2.5 \cdot 10^{-14}$	–	61.1	$2 \cdot 10^{14}$	–	2.59	54	0	54	43	0	54
10^{14}	$5.1 \cdot 10^{-14}$	–	50.4	$2 \cdot 10^{14}$	–	2.45	26	0	26	15	0	26
$2 \cdot 10^{14}$	–	–	–	–	–	–	0	0	0	0	0	0

other results are similar. However, the difference is on a high level: both routines verify some 14 correct figures of the solution. For all accuracy ratios μ_k/μ_1 between 0.5 and 2 the results can be considered to be of similar quality.

Next we come to dimension $n = 1000$ in Table 8.4. Here the results for right hand side `randn(n,1)` are displayed; those for `A * randn(n, 1)` are very similar. Now we display the computing time of Algorithms [II]–[VI] relative to that of Algorithm [I] in columns 8–12. As expected the maximally treatable condition number decreases compared to $n = 100$. Algorithm `LssErrBnd` verifies at least 13 correct digits of the solution, `verifylss` is weaker. Again Theorem 5.2 [VI] is a little weaker, Ogita's method [V] about one decimal digit less accurate, whereas, similar to dimension $n = 100$, Nguyen's method [IV] is superior for condition numbers around $4 \cdot 10^{13}$, but computes a number of zero intervals for larger condition numbers.

The performance of the algorithms depends on the computing time spend to improve the accuracy of the final result. It is not surprising that `LssErrBndNear` is sometimes faster than `LssErrBnd` because all computations are performed in rounding to nearest and no interpretation overhead for operations with interval data types occur. But this happens only for moderately ill-conditioned examples. For well-conditioned matrices the improvements discussed in Section 3.3 accelerate `LssErrBnd` to be faster than `LssErrBndNear`, and for more ill-conditioned examples `LssErrBndNear` fails.

In the design of `LssErrBnd` we compromised between speed and accuracy, where we did not iterate further if an accuracy of about 13 correct decimal digits is achieved. In particular we tried to maintain this compromise over a large range of dimensions and condition numbers. Other methods like Nguyen's [IV] compute better inclusions for certain condition numbers at the price of increasing computing time; for large condition numbers the inclusions are much weaker.

We did not compromise on the applicability of `LssErrBnd` but tried with high priority to compute an inclusion whenever possible. In our examples, the range of applicability of `LssErrBnd` [I], `verifylss` [III] and Theorem 5.2 [VI] is the same, whereas the other methods failed in cases where the aforementioned succeeded. This is indicated in Table 8.4 with an asterisk at the relative computing times in columns 8–12, i.e. the corresponding method was successful in a smaller number of cases than methods [I], [III] and [VI]. The exact percentage of success for the same set of examples as in Table 8.4 is displayed in Table 8.5.

Table 8.6
Results of Algorithm 4.2 (`LssErrBnd`) [I], `verifylss` in INTLAB [III], and Theorem 5.2 [VI], matrix and right hand side with relative radius $r/\mathrm{cond}(A)$, dimension $n = 1000$, r.h.s. `A * randn(n, 1)`, condition number 10^{13}, $K = 100$ test cases.

r	Median (median radius)		Relative comp. time		Percentage of success		
	III/I	VI/I	III/I	VI/I	I	III	VI
$5 \cdot 10^{-16}$	1.01	1.03	0.96	1.05	100	100	100
$6 \cdot 10^{-16}$	1.01	1.03	0.96	1.05	100	100	100
$7 \cdot 10^{-16}$	1.01	1.03	0.96	1.04	100	100	100
$8 \cdot 10^{-16}$	1.01	1.04	0.96	1.05	100	100	100
$9 \cdot 10^{-16}$	1.01	1.04	0.96	1.04	100	100	100
10^{-15}	1.01	1.04	0.96	1.05	100	100	100
$2 \cdot 10^{-15}$	1.01	1.08	0.96	1.04	100	100	100
$3 \cdot 10^{-15}$	1.01	1.16	0.99	1.06	100	100	100
$4 \cdot 10^{-15}$	1.00	1.27	1.01	1.04	100	100	100
$5 \cdot 10^{-15}$	0.99	1.41	1.13	1.05	100	100	100
$6 \cdot 10^{-15}$	0.96	1.55	1.24	1.05	100	100	100
$7 \cdot 10^{-15}$	0.92	1.72	1.30	1.06	85	85	85
$8 \cdot 10^{-15}$	0.91	1.80	1.33	1.07	53	53	53
$9 \cdot 10^{-15}$	0.91	1.82	1.32	1.06	16	16	16

Table 8.7
Results of Algorithm 4.2 (`LssErrBnd`) [I], `verifylss` in INTLAB [III], and Theorem 5.2 [VI], matrix and right hand side with relative radius $r/\mathrm{cond}(A)$, dimension $n = 1000$, r.h.s. `randn(n,1)`, condition number 10^3, $K = 100$ test cases.

r	Median (median radius)		Relative comp. time		Percentage of success		
	III/I	VI/I	III/I	VI/I	I	III	VI
$5 \cdot 10^{-6}$	1.01	1.06	1.15	1.08	100	100	100
$6 \cdot 10^{-6}$	1.00	1.08	1.15	1.07	100	100	100
$7 \cdot 10^{-6}$	1.02	1.11	1.19	1.10	100	100	100
$8 \cdot 10^{-6}$	1.01	1.14	1.18	1.07	100	100	100
$9 \cdot 10^{-6}$	1.01	1.18	1.18	1.08	100	100	100
10^{-5}	1.01	1.22	1.21	1.10	100	100	100
$2 \cdot 10^{-5}$	0.96	1.84	1.71	1.08	95	95	95
$3 \cdot 10^{-5}$	–	–	–	–	0	0	0

Finally we consider data with tolerances. Only `LssErrBnd` [I], `verifylss` [III] and our implementation of Theorem 5.2 [VI] accept input data with tolerances. Again there is not much difference between a right hand side `randn(n,1)` and `A*randn(n)`, so one of them is chosen in the following. Results for linear systems with only the right hand side afflicted with tolerances were already shown in Section 7, in particular Fig. 7.2. Matrices with condition numbers beyond $\mathbf{u}^{-1}$, the data of which is afflicted with tolerances, likely contain a singular matrix. To save space we consider only condition numbers 10^{13} and 10^3.

We first generate a random matrix A of dimension $n = 1000$ with specified condition number and a right hand side $b =$ `randn(n, 1)`. Then, using midpoint-radius notation, algorithms [I], [III] and [VI] are applied to the matrix $\mathbf{A}$ and right hand side $\mathbf{b}$ with $\mathbf{A}_{ij} := \langle A_{ij}, r|A_{ij}|\rangle$ and $\mathbf{b}_j := \langle b_j, r|b_j|\rangle$ for $1 \le i, j \le n$. It means that all entries in $\mathbf{A}$ and $\mathbf{b}$ are afflicted with a relative tolerance r. It can be expected that $\mathbf{A}$ contains singular matrices for $r \gtrsim \mathrm{cond}(A)^{-1}$.

In Table 8.6 results are shown for 100 test cases each with $\mathrm{cond}(A) = 10^{13}$ and different values of r. For μ_k denoting the median over the 100 test cases of the median relative errors of the result of algorithm k, the ratio μ_k/μ_1 for $k \in \{3, 6\}$ is displayed in columns 2 and 3. Next the ratio of computing times between Algorithm k and `LssErrBnd` is shown, and in the last three columns the percentage of success.

As can be seen the range of applicability is the same for all examples. The quality of `verifylss` [III] is better for larger radii requiring more computing time; for small radius better inclusions are achieved in less time. The inclusions by Theorem 5.2 are always weaker than `LssErrBnd` requiring more computing time. This is in particular interesting because of the optimality of the inclusion if there would be no dependencies in $R\mathbf{A}$ as by Theorem 5.1.

Finally we show in Table 8.7 similar results for condition number 10^3 allowing for larger radii. Again the scope of applicability of all three methods Algorithm 4.2 (`LssErrBnd`) [I], `verifylss` in INTLAB [III], and Theorem 5.2 [VI] is the same in the examples. Now `LssErrBnd` delivers almost always the best bounds requiring less computing time than the other methods. The results for other dimensions are similar and therefore omitted.

Note that both for condition number 10^{13} and 10^3 the maximal relative radius for which inclusions are achieved is not too far from $\mathrm{cond}(A)^{-1}$. Also note that success of an algorithm proves all matrices within $\mathbf{A}$ to be non-singular, but failure does not mean that there exists a singular matrix within the tolerances; it means that the problem is too difficult to solve by the given means. It is possible to prove that an interval matrix contains a singular matrix by using some heuristic to "move"

into the right direction within $\mathbf{A}$ to find matrices $A_1, A_2 \in \mathbf{A}$ with $\det(A_1)\det(A_2) \leq 0$. A gap remains where nothing can be said by an efficient algorithm because of the NP-hardness of the problem.

9. Conclusion

In this Part II of the paper directed rounding is used to obtain algorithms with a wider range of applicability. Interval notation increases readability substantially without sacrificing accuracy and/or speed. In particular extremely ill-conditioned linear systems with condition numbers up to $\mathbf{u}^{-2}$ are easily treated by solving the corresponding preconditioned interval linear system. The only additional tool other than directed rounding is the extra-precise accumulation of dot products, which in turn is implemented using solely floating-point operations in working precision.

A new method to compute so-called "inner inclusion" verifies the quality of the computed bounds in case of input data afflicted with tolerances. It is applicable even if only a single entry in the matrix and/or right hand side is afflicted with a tolerance. If tolerances are not too wide, there is not too much difference between the outer and inner inclusion, indicating that the inclusions are almost optimal.

Acknowledgments

My thanks to Florian Bünger and an anonymous referee for helpful comments.

References

[1] S.M. Rump, INTLAB—INTerval LABoratory, in: Tibor Csendes (Ed.), Developments in Reliable Computing, Kluwer Academic Publishers, Dordrecht, 1999, pp. 77–104.
[2] R.B. Kearfott, M. Dawande, C. Hu, Intlib: a portable Fortran-77 interval standard function library, ACM Trans. Math. Software 20 (1994) 447–459.
[3] O. Knüppel, PROFIL/BIAS—a fast interval library, Computing 53 (1994) 277–287.
[4] O. Knüppel, T. Simenec, PROFIL/BIAS extensions, Technical Report 93.5, Forschungsschwerpunkt Informations-und Kommunikationstechnik, Technische Universität Hamburg-Harburg, 1993.
[5] J. Zemke, b4m—BIAS for Matlab, Technical Report, Inst. f. Informatik III, Technische Universität Hamburg-Harburg, 1998.
[6] A. Neumaier, A simple derivation of the Hansen–Bliek–Rohn–Ning–Kearfott enclosure for linear interval equations, Reliab. Comput. 5 (1999) 131–136.
[7] R.A. Horn, Ch. Johnson, Topics in Matrix Analysis, Cambridge University Press, 1991.
[8] Lei Li, On the iterative criterion for generalized diagonally dominant matrices, SIAM J. Matrix Anal. Appl. (SIMAX) 24 (1) (2002) 17–24.
[9] R.S. Varga, Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1962.
[10] J.M. Muller, N. Brisebarre, F. de Dinechin, C.P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, S. Torres, Handbook of Floating-Point Arithmetic, Birkhäuser, Boston, 2010.
[11] ANSI/IEEE 754-1985: IEEE Standard for Binary Floating-Point Arithmetic, New York, 1985.
[12] ANSI/IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic, New York, 2008.
[13] S. Oishi, S.M. Rump, Fast verification of solutions of matrix equations, Numer. Math. 90 (4) (2002) 755–773.
[14] N.J. Higham, Accuracy and Stability of Numerical Algorithms, second ed. SIAM Publications, Philadelphia, 2002.
[15] A. Neumaier, Interval methods for systems of equations, in: Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1990.
[16] S.M. Rump, Fast and parallel interval arithmetic, BIT 39 (3) (1999) 539–560.
[17] R. Krawczyk, Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken, Computing 4 (1969) 187–201.
[18] S.M. Rump, Kleine Fehlerschranken bei Matrixproblemen, Ph.D. Thesis, Universität Karlsruhe, 1980.
[19] G. Hargreaves, Interval analysis in MATLAB, Master's Thesis, University of Manchester, 2002. http://www.manchester.ac.uk/mims/eprints.
[20] E.R. Hansen, Bounding the solution set of interval linear systems, SIAM J. Numer. Anal. (SINUM) 29 (1992) 1493–1503.
[21] C. Bliek, Computer methods for design automation, Ph.D. Dissertation, Massachusetts Institute of Technology MIT, 1992.
[22] J. Rohn, Stability of the optimal basis of a linear program under uncertainty, Oper. Res. Lett. 13 (1993) 9–12.
[23] S. Ning, R.B. Kearfott, A comparison of some methods for solving linear interval equations, SIAM J. Numer. Anal. (SINUM) 34 (4) (1997) 1289–1305.
[24] A. Neumaier, Erratum to: a simple derivation of the Hansen–Bliek–Rohn–Ning–Kearfott enclosure for linear interval equations, Reliab. Comput. 6 (1999) 227.
[25] A. Neumaier, private communication, 2010.
[26] S. Poljak, J. Rohn, Checking robust nonsingularity is NP-hard, Math. Control Signals Systems 6 (1993) 1–9.
[27] V. Kreinovich, A.V. Lakeyev, S.I. Noskov, Optimal solution of interval linear systems is intractable (NP-hard), Interval Comput. 1 (1993) 6–14.
[28] J. Rohn, Linear interval equations: computing enclosures with bounded relative or absolute overestimation is NP-hard, in: R.B. Kearfott, V. Kreinovich (Eds.), Applications of Interval Computations, Kluwer Academic Publisher, 1991, pp. 81–89.
[29] G.E. Coxson, Computing exact bounds on elements of an inverse interval matrix is NP-hard, Reliab. Comput. 5 (2) (1999) 137–142.
[30] T. Ogita, S. Oishi, Y. Ushiro, Fast inclusion and residual iteration for solutions of matrix equations, Comput. Suppl. 16 (2002) 171–184.
[31] T. Ogita, S. Oishi, Y. Ushiro, Fast Verification of Solutions for Sparse Monotone Matrix Equations, in: Comput. Suppl., vol. 15, Springer, Wien, 2001, pp. 175–187.
[32] H.D. Nguyen, N. Revol, Accuracy issues in linear algebra using interval arithmetic, SCAN Conference Lyon, 2010.
[33] H.D. Nguyen, Efficient algorithms for verified scientific computing: numerical linear algebra using interval arithmetic, Dissertation, Laboratoire LIP, INRIA ENS Lyon, 2011.
[34] S.M. Rump, Verification methods for dense and sparse systems of equations, in: J. Herzberger (Ed.), Topics in Validated Computations—Studies in Computational Mathematics, Elsevier, Amsterdam, 1994, pp. 63–136.
[35] A. Neumaier, Overestimation in linear interval equations, SIAM J. Numer. Anal. (SINUM) 24 (1) (1987) 207–214.
[36] A. Neumaier, Rigorous sensitivity analysis for parameter-dependent systems of equations, J. Math. Anal. Appl. 144 (1989).
[37] S.M. Rump, Rigorous sensitivity analysis for systems of linear and nonlinear equations, Math. Comp. 54 (10) (1990) 721–736.
[38] J. Rohn, VERSOFT: verification software in MATLAB/INTLAB, 2009.
[39] R. Skeel, Iterative refinement implies numerical stability for Gaussian elimination, Math. Comp. 35 (151) (1980) 817–832.