Developing robust incomplete Cholesky factorizations in half precision arithmetic

Numerical Algorithms
https://doi.org/10.1007/s11075-025-02015-x

RESEARCH

Developing robust incomplete Cholesky factorizations in half precision arithmetic

Jennifer Scott[1,2] · **Miroslav Tůma**[3]

Received: 24 April 2024 / Accepted: 17 January 2025

Abstract

Incomplete factorizations have long been popular general-purpose algebraic preconditioners for solving large sparse linear systems of equations. Guaranteeing the factorization is breakdown free while computing a high quality preconditioner is challenging. A resurgence of interest in using low precision arithmetic makes the search for robustness more important and more challenging. In this paper, we focus on ill-conditioned symmetric positive definite problems and explore a number of approaches for preventing and handling breakdowns: prescaling of the system matrix, a look-ahead strategy to anticipate breakdown as early as possible, the use of global shifts, and a modification of an idea developed in the field of numerical optimization for the complete Cholesky factorization of dense matrices. Our numerical simulations target highly ill-conditioned sparse linear systems with the goal of computing the factors in half precision arithmetic and then achieving double precision accuracy using mixed precision refinement. We also consider the often overlooked issue of growth in the sizes of entries in the factors that can occur when using any precision and can render the computed factors ineffective as preconditioners.

Keywords Half precision arithmetic · Preconditioning · Incomplete factorizations · Iterative methods for linear systems

Jennifer Scott and Miroslav Tůma contributed equally to this work.

✉ Jennifer Scott
jennifer.scott@reading.ac.uk

Miroslav Tůma
mirektuma@karlin.mff.cuni.cz

1 School of Mathematical, Physical and Computational Sciences, University of Reading, Reading RG6 6AQ, UK

2 STFC Rutherford Appleton Laboratory, Didcot, Oxfordshire OX11 0QX, UK

3 Department of Numerical Mathematics, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

1 Introduction

Our interest is in solving large-scale symmetric positive definite (SPD) linear systems of equations $Ax = b$. Incomplete Cholesky (IC) factorizations of the form $A \approx LL^T$, where the factor L is a sparse lower triangular matrix, have long been important and well-used algebraic preconditioners for use with iterative solvers. While they are general purpose and, when compared to sparse direct solvers, require modest computational resources, they do have drawbacks. Their effectiveness can be highly application dependent and, although significant effort has gone into developing a strong theoretical background, most results are limited to model problems. To be useful in practice, their computation and application must be efficient and robust.

Traditionally, matrix factorizations have most often been computed using double precision floating-point arithmetic, which nowadays corresponds to a 64-bit floating-point number format. However, half precision arithmetic is being increasingly supported by modern hardware and because it can offer speed benefits while using less energy and memory, there has been significant interest in recent years in its use in numerical linear algebra; see the comprehensive review [1] and references therein. For linear systems, one strategy that has received attention is GMRES-IR [2]. The idea is to compute the matrix factors in low precision arithmetic and then employ them as preconditioners for GMRES within mixed precision iterative refinement (see also [1, 3, 4]). For SPD systems, GMRES can potentially be replaced by the CG (conjugate gradient) method [5]. Using low precision incomplete factors may enable much larger problems to be solved (normally at the cost of more iterations to achieve the requested accuracy). In an initial study [6], we explored this approach, focusing on the safe avoidance of overflows that can occur when computing matrix factors using low precision arithmetic.

When using any precision, breakdown can occur during an incomplete Cholesky factorization of a general SPD matrix, that is, a pivot (diagonal entry of a Schur complement) may be zero or negative where an exact Cholesky factorization would have only positive pivots, or a computation within the factorization may overflow. A review is given in the Lapack Working Note [7]. Breakdown is more likely when using low precision arithmetic. This is partly because the initial matrix A must be "squeezed" into half precision, which may mean the resulting matrix is not (sufficiently) positive definite for the factorization to be successful [8]. But, in addition, overflows outside the narrow range of possible numerical values are an ever-present danger. Our interest lies in exploring strategies to prevent breakdown during sparse matrix factorizations. Importantly, these must be robust, inexpensive and not cause serious degradation to the preconditioner quality.

Building on our earlier work on breakdowns within half precision sparse matrix incomplete factorizations [6], this paper makes the following contributions. Firstly, we consider and compare the performance in half precision arithmetic of a number of strategies to limit the likelihood of breakdown: (a) prescaling the matrix before it is squeezed, (b) look-ahead that checks the diagonal entries of the partially factorized matrix at each major step of the factorization, (c) the use of global shifts, and (d) an approach based on locally modifying the factorization. The latter was originally used to modify approximate (dense) Hessian matrices in the field of numerical optimization. Here we seek to apply it to sparse problems, in combination with low precision arithmetic, as a strategy that avoids the restarting needed with the use of global shifts. Secondly, we demonstrate that, even if double precision is used throughout the computation and breakdown does not occur, it is essential to take action to prevent growth in the factor entries because otherwise, the factors can be ineffective as preconditioners. Finally, we concentrate our numerical experiments on highly ill-conditioned linear systems and the challenge of recovering double accuracy in the computed solution using a preconditioner computed in low precision arithmetic. We develop Fortran software that enables us to illustrate the potential for exploiting half precision within robust approaches for tackling large-scale sparse systems.

The rest of the paper is organised as follows. Section 2 looks at the different stages at which breakdown can occur within an incomplete factorization. In Section 3, we present a number of ways to prevent and handle breakdown. Numerical results for a range of highly ill-conditioned linear systems coming from practical applications are presented in Section 4. Finally, in Section 5, our findings and conclusions are summarised.

Terminology We use high precision (denoted by fp64) to refer to IEEE double precision (64-bit) and low precision (denoted by fp16) for the 1985 IEEE standard 754 half precision (16-bit). Note that bfloat16 is another form of half precision arithmetic. It has 8 bits in the significand and (as in fp32) 8 bits in the exponent. We do not use it in this paper because our software is written in Fortran and, as far as we are aware, there are currently no Fortran compilers that support the use of bfloat16. Table 1 summarises the parameters for the precisions used in this paper.

2 Possible breakdowns within IC factorizations

A myriad of approaches for computing incomplete factorizations of sparse matrices have been developed, modified and refined over many years. Some combine the factorization with an initial step that discards small entries in A (sparsification). For details of possible variants, we recommend [9, 10], while a comprehensive discussion of early strategies can be found in [11]; see also [12] for a short history and the recent monograph [13] for a broad overview and skeleton algorithms. We note that significant

Table 1 Parameters for fp16, fp32, and fp64 arithmetic: the number of bits in the significand (including the implicit most significant bit) and exponent, unit roundoff u, smallest positive (subnormal) number x^s_{min}, smallest normalized positive number x_{min}, and largest finite number x_{max}, all given to three significant figures

	Signif.	Exp.	u	x^s_{min}	x_{min}	x_{max}
fp16	11	5	4.88×10^{-4}	5.96×10^{-8}	6.10×10^{-5}	6.55×10^{4}
fp32	24	8	5.96×10^{-8}	1.40×10^{-45}	1.18×10^{-38}	3.40×10^{38}
fp64	53	11	1.11×10^{-16}	4.94×10^{-324}	2.22×10^{-308}	1.80×10^{308}

progress in the field of preconditioning has been achieved by looking for incomplete factorizations that are breakdown-free because of the properties of the matrix A, for example, for M and H-matrices. A seminal paper on this is [14]; see also the summary in [15].

Algorithm 1 outlines a basic (right-looking) incomplete Cholesky (IC) factorization of a sparse SPD matrix $A = \{a_{ij}\}$.[1] It assumes a target sparsity pattern $\mathcal{S}\{L\}$ for the incomplete factor $L = \{l_{ij}\}$ is provided, where

$$\mathcal{S}\{L\} = \{(i, j) \mid l_{ij} \neq 0,\ 1 \leq j \leq i \leq n\}.$$

The simplest case $\mathcal{S}\{L\} = \mathcal{S}\{A\}$ is called an $IC(0)$ factorization. Modifications to Algorithm 1 can be made to incorporate threshold dropping strategies and to determine $\mathcal{S}\{L\}$ as the method proceeds. At each major step k, outer product updates are applied to the part of the matrix that has yet to be factored (Lines 7–11).

Algorithm 1 Basic right-looking sparse IC factorization.

Input: Sparse SPD matrix A and a target sparsity pattern $\mathcal{S}\{L\}$
Output: Incomplete Cholesky factorization $A \approx LL^T$

1: $l_{ij} = a_{ij}$ for all $(i, j) \in \mathcal{S}\{L\}$
2: **for** $k = 1 : n$ **do** ▷ Start of k-th major step
3:     $l_{kk} \leftarrow (l_{kk})^{1/2}$ ▷ Diagonal entry is the pivot
4:     **for** $i \in \{i > k \mid (i, k) \in \mathcal{S}\{L\}\}$ **do**
5:         $l_{ik} \leftarrow l_{ik}/l_{kk}$ ▷ Scale pivot column k of the incomplete factor by the pivot
6:     **end for** ▷ Column k of L has been computed
7:     **for** $j \in \{j > k \mid (j, k) \in \mathcal{S}\{L\}\}$ **do**
8:         **for** $i \in \{i \geq k \mid (i, k) \in \mathcal{S}\{L\}\}$ **do**
9:             $l_{ij} \leftarrow l_{ij} - l_{ik} l_{jk}$ ▷ Update operation on column $j > k$
10:         **end for**
11:     **end for**
12: **end for**

Unfortunately, unlike a complete Cholesky factorization, there is no guarantee in general that an IC algorithm will not break down or exhibit large growth in the size of the factor entries (even when using double precision arithmetic). This is illustrated

[1] The Algorithm can be modified to compute a square-root free LDLT factorization in which L has unit diagonal entries and D has positive entries.

by the following well-conditioned SPD matrix in which $\delta > 0$ is small

$$A = \begin{pmatrix} 3 & -2 & 0 & 2 & 0 \\ -2 & 3 & -2 & c & 0 \\ 0 & -2 & 3 & -2 & 0 \\ 2 & c & -2 & 8+2\delta & 2 \\ 0 & 0 & 0 & 2 & 8 \end{pmatrix}.$$

Choosing $\delta \ll 1$ and $c = 1$ results in no growth in the entries of the $IC(0)$ factor and no breakdown. However, if $c = 0$ and entries (2,4) and (4,2) are removed from $\mathcal{S}\{A\}$ then the $IC(0)$ factor becomes

$$\begin{pmatrix} d_1 & & & & \\ -2/d_1 & d_2 & & & \\ 0 & -2/d_2 & d_3 & & \\ 2/d_1 & 0 & -2/d_3 & d_4 & \\ 0 & 0 & 0 & 2/d_4 & d_5 \end{pmatrix},$$

with $d_1^2 = 3$, $d_2^2 = 5/3$, $d_3^2 = 3/5$, $d_4^2 = 2\delta$, and $d_5^2 = 8 - 2/\delta$. In this case, if $\delta \ll 1$ then there is large growth in the (5,4) entry and the factorization breaks down because the (5,5) entry is negative (for any working precision).

There are three places in Algorithm 1 where breakdown can occur. Following [6], we refer to these as B1, B2, and B3 breakdowns.

- B1: The diagonal entry l_{kk} may be unacceptably small or negative.
- B2: The column scaling $l_{ik} \leftarrow l_{ik}/l_{kk}$ may overflow.
- B3: The update operation $l_{ij} \leftarrow l_{ij} - l_{ik}l_{jk}$ may overflow.

To develop robust IC factorization implementations, breakdowns must either be avoided or they must be detected and handled by restarting the computation with revised data. We seek to avoid breakdowns but, as we cannot guarantee there will be no breakdowns, we still need to monitor for them. A recent study involving multi-precision iterative refinement used the functions offered by MATLAB to check the computed factors for Inf and/or NaN entries and took action if such entries were found [16]. This is not a practical procedure for general use. One possible strategy is to use IEEE-754 floating-point exception handling. This allows overflows to occur, the execution continues until a status flag is checked and, at this point, if overflow has been detected, restarting is initiated; see, for example, [17]. This is straightforward but requires the user to employ the correct compiler flags, which may be challenging, for instance, when a solver is interfaced from other languages. Furthermore, it is likely that non-IEEE arithmetics will gain traction in the future [18]. An alternative and potentially more flexible strategy is to incorporate explicit tests for breakdown into the factorization algorithm. In this case, for an implementation to be robust, the tests employed must only use operations that cannot themselves overflow.

An operation is said to be *safe* in the precision being used if it cannot overflow. To safely detect B1 breakdown it is sufficient to check at Line 3 of Algorithm 1 that $l_{kk} \geq \tau_u$, where the threshold parameter satisfies $\tau_u > 1/x_{max}$. This ensures

$(l_{kk})^{-1} < x_{max}$. Typical values are $\tau_u = 10^{-5}$ for half precision factorizations and $\tau_u = 10^{-20}$ for double precision [6]; these are used in our reported experiments (Section 4). B2 breakdown can happen at Line 5. Let l_{kmax} denote the entry below the diagonal in column k of largest absolute value, that is,

$$l_{kmax} = \max_{i>k} \{|l_{ik}| : (i,k) \in \mathcal{S}\{L\}\}. \tag{2.1}$$

If $l_{kmax} \le x_{max}$ and $1 \le l_{kk} \le x_{max}$ or $l_{kk} \ge l_{kmax}/x_{max}$ then it is safe to compute l_{kmax}/l_{kk} (and thus safe to scale column k). B3 breakdown can occur at Line 9. We give an algorithm for safely detecting B3 breakdown in [6]. Given scalars a, b, c such that $|a|, |b|, |c| \le x_{max}$, the algorithm returns $v = a - bc$ or a flag to indicate v cannot be computed safely. It does this in two stages: it first checks whether $w = bc$ can be computed safely and then whether $v = a - w$ can be computed safely.

Note that although we are focusing on SPD problems and IC factorizations, B1, B2 and B3 breakdowns are also possible during complete or incomplete factorizations of nonsymmetric sparse matrices. Indeed, for non SPD problems, B2 breakdowns (that is, overflow of one or more entries when the pivot column is divided by the pivot) in particular may be more likely to occur (although remains uncommon if the matrix is well-scaled). To demonstrate how B2 breakdown can happen, consider the following nonsymmetric matrix, which has some large off-diagonal entries

$$A = \begin{pmatrix} 3 & -2 & 0 & 2 & 2 \\ -2 & 3 & -2 & 0 & 0 \\ 0 & -2 & 3 & -1 & -1 \\ 2 & 0 & -2 & 2.01 & 2.01 \\ 1000 & 1000 & 1000 & 1000 & 100 \end{pmatrix}.$$

The LU factorization of A is given by

$$A = LU = \begin{pmatrix} 1 & & & & \\ -2/3 & 1 & & & \\ 0 & -1.2 & 1 & & \\ 2/3 & 0.8 & -2/3 & 1 & \\ 1000/3 & 1000 & 5000 & -400000 & 1 \end{pmatrix} \begin{pmatrix} 3 & -2 & 2 & 0 & 2 \\ & 5/3 & -2 & 4/3 & 4/3 \\ & & 0.6 & 0.6 & 0.6 \\ & & & 0.01 & 0.01 \\ & & & & -900 \end{pmatrix}.$$

When using fp16 arithmetic, B2 breakdown occurs when performing the 4th elimination step because the (5, 4) entry overflows (-4000 is divided by 0.01).

3 Preventing and handling breakdown in IC factorizations

While the use of safe tests allows action to be taken before breakdown occurs or the use of IEEE exception handling can capture breakdown, our objective is to reduce the likelihood of breakdown. This will limit the overheads involved in handling breakdowns and the effects on the quality of the computed factorizations through modifications to the data. Breakdown is much more likely to happen when using low precision arithmetic because of the greater likelihood of overflows occurring.

3.1 Avoiding breakdown by prescaling

For both direct and iterative methods for solving systems of equations it is often beneficial to prescale the matrix, that is, to determine diagonal matrices S_r and S_l (with $S_r = S_l$ in the symmetric case) such that the scaled matrix $\widehat{A} = S_r^{-1} A S_l^{-1}$ is "nicer" than the original A. By nicer, we mean that, compared with solving $Ax = b$, it is easier to solve the system $\widehat{A}y = S_r^{-1}b$ and then set $x = S_l^{-1}y$. When working in fp16 arithmetic, scaling is essential because of the narrow range of the arithmetic (recall Table 1). Numbers of absolute value outside the interval $[x_{min}^s, x_{max}] = [5.96 \times 10^{-8}, 6.55 \times 10^4]$ cannot be represented in fp16 arithmetic and they underflow or overflow when converted to fp16 arithmetic. Moreover, to avoid the performance penalty of handling subnormal numbers, in practice numbers with small absolute values are often flushed to zero (that is, replaced by zero). Before factorizing $\widehat{A}$ in fp16 arithmetic, a scaling is chosen so that when converting (squeezing) the scaled matrix into fp16, overflow is avoided. In this initial squeezing of the matrix, we flush to zero all entries in the scaled matrix of absolute value less than 10^{-5}. For incomplete factorizations, numbers that underflow or are flushed to zero are not necessarily a concern because the factorization is approximate. However, the resulting sparsification may mean that the scaled and squeezed matrix is close to being indefinite.

No single approach to constructing a scaling is universally the best and sparse solvers frequently include a number of options to allow users to experiment to determine the most effective for their applications (or to supply their own scaling). Our experience with IC factorizations of SPD matrices is that it is normally sufficient to use simple l_2-norm scaling (that is, using fp64 arithmetic, we compute $S_r = S_l = D^{1/2}$, where d_{ii} is the 2-norm of row i of A), resulting in the absolute values of the entries of the scaled matrix $\widehat{A}$ being at most 1. This is used in the current study (but see [19], where equilibration scaling is used and [5] where scaling by the square root of the diagonal is used).

3.2 Preventing breakdown by incorporating look-ahead

Recall that the computation of the diagonal entries of the factor in a (complete or incomplete) Cholesky factorization are based on

$$l_{jj} = a_{jj} - \sum_{i<j} l_{ij}^2.$$

Initially, $l_{jj} = a_{jj}$ and at each stage of the factorization a positive (or zero) term is subtracted from it so that l_{jj} either decreases or remains the same on each major step

k. Thus, to detect potential B1 breakdown as early as possible, look-ahead can be used whereby at each step k, the remaining diagonal entries l_{jj} $(j > k)$ are updated (using safe operations) and tested. For the right-looking Algorithm 1, it is straightforward to incorporate testing but, for some IC variants, it may be necessary to hold a copy of the diagonal entries of the factor. Look-ahead is employed in some well-known fp64 arithmetic implementations of IC factorizations e.g., [20, 21]. Algorithm 2 is a modified version of Algorithm 1 that includes checks for breakdown. The safe test for B3 breakdown can be modified so that, ahead of the loop at Line 13, a check is made that the entry of maximum magnitude in column k (that is, l_{kmax} from (2.1)) is less than $(x_{max})^{1/2}$. If it is not, $flag = 3$ is returned. Otherwise, the multiplication of l_{ik} and l_{jk} is safe and, at Line 17, only the subtraction needs to be checked. If in place of the safe tests, IEEE exception handling is used then the IEEE overflow flag should be tested at the end of each major loop (that is, between Lines 20 and 21).

Algorithm 2 Right-looking IC factorization with safe checks for breakdown.

Input: SPD matrix A, a target sparsity pattern $\mathcal{S}\{L\}$, parameter $\tau_u > 0$
Output: Either $flag = 0$ and $A \approx LL^T$ or $flag > 0$ (breakdown detected)

1: $l_{ij} = a_{ij}$ for all $(i, j) \in \mathcal{S}\{L\}$
2: $flag = 0$
3: **if** $l_{11} < \tau_u$ **then** $flag = 1$ and **return** ▷ B1 breakdown
4: **for** $k = 1 : n$ **do** ▷ Start of k-th major step
5: $\quad l_{kk} \leftarrow (l_{kk})^{1/2}$
6: $\quad$ **if** $l_{kk} \geq 1$ or $l_{kk} \geq l_{max}/x_{max}$ **then** ▷ l_{max} is largest off-diagonal entry (2.1)
7: $\quad\quad$ **for** $i \in \{i > k \mid (i, k) \in \mathcal{S}\{L\}\}$ **do**
8: $\quad\quad\quad l_{ik} \leftarrow l_{ik}/l_{kk}$ ▷ Perform safe scaling
9: $\quad\quad$ **end for** ▷ Column k of L has been computed
10: $\quad$ **else**
11: $\quad\quad flag = 2$ and **return** ▷ B2 breakdown
12: $\quad$ **end if**
13: $\quad$ **for** $j \in \{j > k \mid (j, k) \in \mathcal{S}\{L\}\}$ **do** ▷ Update columns $j > k$ of L
14: $\quad\quad$ **for** $i \in \{i \geq j \mid (i, j) \in \mathcal{S}\{L\}\}$ **do**
15: $\quad\quad\quad$ Test entry (i, j) can be updated safely ▷ Use Algorithm 2.2 of [6]
16: $\quad\quad\quad$ **if** not safe to update **then** $flag = 3$ and **return** ▷ B3 breakdown
17: $\quad\quad\quad l_{ij} \leftarrow l_{ij} - l_{ik} l_{jk}$ ▷ Perform safe update operation
18: $\quad\quad$ **end for**
19: $\quad\quad$ **if** $l_{ii} < \tau_u$ **then** $flag = 1$ and **return** ▷ B1 breakdown
20: $\quad$ **end for**
21: **end for**

To see the usefulness of look-ahead, consider the following matrix

$$A = \begin{pmatrix} 3 & -2 & 0 & 1 & 2 \\ -2 & 3 & -2 & 0 & 0 \\ 0 & -2 & 3 & 0 & -2 \\ 1 & 0 & 0 & 5 & 0 \\ 2 & 0 & -2 & 0 & 8 \end{pmatrix}.$$

It is easy to check that A is SPD with condition number $\kappa_2(A) \approx 2 \times 10^6$. If the $IC(0)$ factorization is computed in exact arithmetic then the entry (5, 5) of L is zero (B1 breakdown). The look-ahead strategy reveals this at the third step, thus reducing the work performed before breakdown is detected.

A consequence of look-ahead is that, through the early detection of B1 breakdowns and taking action to prevent such breakdowns, B3 breakdowns are indirectly prevented. In our numerical experiments on problems coming from real applications, all breakdowns when using fp16 arithmetic were of type B1 when look-ahead was incorporated. However, B2 and B3 breakdowns remain possible. Consider the following well-conditioned SPD matrix and its complete Cholesky factor (three decimal places):

$$A = \begin{pmatrix} 3 & -2 & 0 & 2 & 0 \\ -2 & 3 & -2 & 0 & 0 \\ 0 & -2 & 3 & -2 & 0 \\ 2 & 0 & -2 & 8.00007 & 550 \\ 0 & 0 & 0 & 550 & 60000 \end{pmatrix}, \quad L = \begin{pmatrix} 1.732 & & & & \\ -1.155 & 1.291 & & & \\ 0 & -1.549 & 0.775 & & \\ 1.155 & 1.033 & -0.516 & 2.309 & \\ 0 & 0 & 0 & 95.254 & 30.414 \end{pmatrix}.$$

Observe that the (4, 2) entry has filled in. The $IC(0)$ factorization does not allow fill-in and, after four steps, the first four columns of the $IC(0)$ factor of A are given by

$$L_{1:5,1:4} = \begin{pmatrix} 1.732 & & & \\ -1.155 & 1.291 & & \\ 0 & -1.549 & 0.777 & \\ 1.155 & 0 & -2.582 & 0.008 \\ 0 & 0 & 0 & 65738 \end{pmatrix}.$$

In fp16 arithmetic, the (5, 4) entry overflows (B3 breakdown). This happens even with look-ahead because the first three entries in row 5 of A are zero and so the (5, 4) entry is not updated until after column 4 has been computed. In this example, the (4, 4) entry before its square root is taken is greater than τ_u (it is equal to 7×10^{-5}) and so there is no B1 breakdown in column 4.

3.3 Global shifting to handle breakdown

Once potential breakdown has been detected (either through the B1-B3 tests or using IEEE exception handling) and the factorization halted, a common approach is to modify all the diagonal entries by selecting $\alpha > 0$, replacing the scaled matrix $\widehat{A}$ by $\widehat{A}+\alpha I$

and restarting the factorization. In exact arithmetic, there is always an α^* such that for all $\alpha \geq \alpha^*$ the IC factorization of $\widehat{A} + \alpha I$ exists [22]. In practice, α^* is unlikely to be known a priori and it may be necessary to restart the factorization a number of times with ever larger shifts. If α was to reach x_{max} in the working precision u then there would have been large growth in the factor entries and it would be necessary to restart using higher precision. However, this would be unlikely to result in a useful preconditioner and α growing in this way was not observed in any of our tests (including highly ill-conditioned examples). Algorithm 3 summarizes the global shifting strategy for a SPD matrix held in precision u and for which an IC factorization in precision $u_l \geq u$ is wanted. After each unsuccessful factorization attempt, the shift is doubled [20]. Here $A_l = fl_l(\widehat{A})$ denotes converting the matrix $\widehat{A}$ from precision u to precision u_l. In practice, it is unnecessary to explicitly hold A_l. Instead, entries of $\widehat{A}$ are cast to precision u_l on the fly as needed.

Based on our experience with a range of problems, in our reported tests, the initial shift is taken to be $\alpha_S = 10^{-3}$. The precise choice of the shift is not critical but it should not be unnecessarily large as this may result in the computed factors providing poor quality preconditioners. Note that more sophisticated strategies for changing the shift, which also allow the possibility for a shift to be decreased, are possible [21]. These have been developed for fp64 arithmetic. Our initial experiments suggest it is less clear that there are significant benefits of doing this when using fp16 arithmetic so in our experiments we only report results for using the simple shift doubling strategy.

Algorithm 3 Shifted incomplete IC factorization in precision u_l.

Input: SPD matrix A in precision u, diagonal scaling matrix S, a target sparsity pattern $\mathcal{S}\{L\}$, and initial shift $\alpha_S > 0$

Output: Shift $\alpha \geq 0$ and incomplete Cholesky factorization $S^{-1}AS^{-1} + \alpha I \approx LL^T$ in precision u_l.

```
1: Â = S^{-1} A S^{-1}                        ▷ Symmetrically scale A
2: A_l = fl_l(Â)                              ▷ Convert to precision u_l
3: α_0 = 0
4: for k = 0, 1, 2, ... do
5:     A_l + α_k I ≈ LL^T in precision u_l     ▷ We use Algorithm 4
6:     If successful then set α = α_k and return
7:     α_{k+1} = max(2α_k, α_S)
8: end for
```

3.4 Local modifications to prevent breakdown

The next strategy is based on seeking to guarantee that the factorization exists by bounding the off-diagonal entries in L. So-called modified Cholesky factorization schemes have been widely used in nonlinear optimization to compute Newton-like directions. Given a symmetric (and possibly indefinite) A, a modified Cholesky algorithm factorizes $A + A_E$, where A_E is termed the correction matrix. The objectives are to compute the correction at minimal additional cost and to ensure $A + A_E$ is SPD and well-conditioned and close to A. A stable approach for dense matrices was originally proposed by Gill and Murray [23] and was subsequently refined and used by Gill, Murray and Wright (GMW) [24] and others [25–29].

Our GMW variant allows for sparse A and incomplete factorizations. In particular, the incompleteness that can lead to significant growth in the factor is a feature that implies modification of the GMW strategy is necessary. Consider the second example in Section 3.2. It clearly shows that once a diagonal entry is small, but not smaller than τ_u, it may be difficult to get a useful factorization by increasing this diagonal entry by an initially prescribed value that is independent of other entries in its column. In the GMW approach, at the start of major step k of the factorization algorithm, the updated diagonal entry l_{kk} is checked (before its square root is taken). If it is too small compared to the off-diagonal entries in its column then it is modified; a parameter $\beta > 0$ controls the local modification. Specifically, at step k, we set

$$l_{kk} = \max\left\{ l_{kk},\ \left(\frac{l_{kmax}}{\beta}\right)^2 \right\}, \tag{3.1}$$

where l_{kmax} is given by (2.1). If $(l_{kmax}/\beta)^2$ overflows (this can be safely checked) then the diagonal entry cannot be modified in this way. We call this a B4 breakdown. If despite the local modification potential breakdown is detected then the factorization is terminated and restarted using a global shift. As the following result shows, the GMW(β) strategy limits the size of the off-diagonal entries in L and, for β sufficiently small, it prevents B3 breakdown.

Lemma 1 *Let the matrix A be sparse and SPD. Assume that, using the GMW(β) strategy, columns 1 to $j-1$ columns of the IC factor L have been successfully computed in fp16 arithmetic. For $i \geq j$ let $nz(i)$ denote the number of nonzero entries in $L_{i,1:j-1}$. If*

$$|a_{ij}| + \min(nz(i), nz(j))\beta^2 \leq x_{max} \quad \textit{for all} \quad (i, j) \in \mathcal{S}\{L\}, \tag{3.2}$$

where x_{max} is the largest finite number represented in fp16, then B3 breakdown cannot occur in the j-th step.

Proof From (3.1), the off-diagonal entries in the first $j-1$ columns of L satisfy

$$\frac{|l_{ik}|}{(l_{kk})^{1/2}} \leq \frac{|l_{ik}|\beta}{l_{kmax}} \leq \beta, \quad 1 \leq k \leq j-1,\ i > k.$$

For any $i > j$ we have

$$l_{ij} = \frac{1}{(l_{jj})^{1/2}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) := \frac{\widetilde{l}_{ij}}{(l_{jj})^{1/2}}. \tag{3.3}$$

To avoid breakdown, we require $|\widetilde{l}_{ij}| \leq x_{max}$. From (3.2) and (3.3),

$$|\widetilde{l}_{ij}| \leq |a_{ij}| + \min(nz(i), nz(j))\beta^2 \leq x_{max},$$

and hence B3 breakdown does not occur. □

Rules to determine the parameter β for the complete Cholesky factorization of dense matrices (in double precision) are discussed in [23, 24] but for sparse incomplete factorizations in fp16 arithmetic such sophisticated rules are not applicable. Provided β is not very small, its value is not critical to the quality of the preconditioner. Given β, Algorithm 4 incorporates the use of the GMW(β) strategy within the incomplete factorization. Note that the local modifications are not combined with look-ahead. As in Algorithm 2, the cost of checking for breakdowns is small. The most expensive step is computing l_{max}.

Algorithm 4 Right-looking IC factorization with safe checks for breakdown and GMW local modifications.

Input: SPD matrix A, a target sparsity pattern $\mathcal{S}\{L\}$, parameters $\tau_u > 0$ and $\beta > 0$
Output: Either $flag = 0$ and $A \approx LL^T$ or $flag > 0$ (breakdown detected)

$l_{ij} = a_{ij}$ for all $(i, j) \in \mathcal{S}\{L\}$
Set $flag = 0$
for $k = 1 : n$ **do** ▷ Start of k-th major step
  if $(l_{max}/\beta)^2$ does not overflow **then** ▷ l_{max} is largest off-diagonal entry (2.1)
    Set $l_{kk} = \max\left\{l_{kk},\ (l_{max}/\beta)^2\right\}$.
  else
    $flag = 4$ and **return** ▷ B4 breakdown
  end if
  if $l_{kk} < \tau_u$ **then** $flag = 1$ and **return** ▷ B1 breakdown
  Follow Lines 5–22 of Algorithm 2, with Lines 18–20 (look-ahead) removed.
end for

3.5 Recovering double precision accuracy

Having computed an incomplete factorization in low precision, we seek to recover (close to) double precision accuracy in the final solution (although in many applications much less accuracy may be sufficient and may be all that is justified by the accuracy in the data). In their work on using mixed precision for solving general linear systems, Carson and Higham [2] introduce a variant of iterative refinement that uses GMRES preconditioned by the low precision LU factors of the matrix to solve the correction equation (GMRES-IR). Carson and Higham employ two precisions. This was later extended to three precisions and then to five precisions [3, 4]; see Algorithm 5, where we use a generic Krylov solver and the low precision incomplete factors. Here u is the working precision. In the three-precision variant [4], $u_p = u$, $u_p = u^2$ and $u_l \geq u \geq u_r$ and typical combinations include $(u_l, u, u_r) = (u_{16}, u_{32}, u_{64})$ or (u_{16}, u_{64}, u_{64}). Section 3.4 of [3] discusses meaningful combinations of the five precisions. They must satisfy $u^2 \leq u_r \leq u \leq u_l$, $u_p \leq u_g$ and $u_p < u_l$. The use of more than two precisions has the potential to solve problems that are less well conditioned in less time and using less memory.

In the SPD case, a natural choice is to select the conjugate gradient (CG) method to be the Krylov solver. The supporting rounding error analysis for GMRES-IR relies on the backward stability of GMRES and preconditioned CG is not guaranteed to be backward stable [30]. Nevertheless, mixed precision results presented in [5] (using MATLAB code and relatively small test examples) suggest that in practice CG-IR can perform as well as GMRES-IR. In our earlier paper [6], we experiment with IC-CG-IR and compare it with IC-GMRES-IR using fp16 and fp64 arithemtic. While there is little to choose between them when run on well-conditioned SPD problems, for highly ill-conditioned examples, IC-CG-IR often (but not always) requires a greater number of iterations to obtain double precision accuracy.

Algorithm 5 IC-Krylov-IR: Krylov solver iterative refinement using five precisions.

Input: SPD matrix A and vector b in precision u, five precisions u_r, u_g, u_p, u and u_ℓ, maximum number of outer iterations $itmax > 0$

Output: Computed solution of the system $Ax = b$ in precision u

1: Compute IC factorization of A in precision u_ℓ
2: Initialize $x_1 = 0$
3: **for** $i = 1 : itmax$ or until convergence **do**
4: Compute $r_i = b - Ax_i$ in precision u_r; store r_i in precision u
5: Use a preconditioned Krylov solver to solve $Ad_i = r_i$ at precision u_g, with preconditioning and products with A in precision u_p; store d_i in precision u
6: Compute $x_{i+1} = x_i + d_i$ in precision u
7: **end for**

4 Numerical experiments

We follow a number of others working on the development of numerical linear algebra algorithms in mixed precision in performing experiments that aim to explore the feasibility of the ideas by using half precision (see, for example, [5, 8, 31, 32]). We want the option to experiment with sparse problems that may be too large for MATLAB and have chosen to develop our software in Fortran. We use the NAG compiler (Version 7.1, Build 7118). As far as we know, it is the only multi-platform Fortran compiler that currently fully supports the use of fp16 arithmetic and conforms to the IEEE standard. In addition, using the `-roundhreal` option, all half-precision operations are rounded to half precision, both at compile time and runtime. Our numerical experiments are performed on a Windows 11-Pro-based machine with an Intel(R) Core(TM) i5-10505 CPU processor (3.20 GHz).

Our test set of SPD matrices is given in Table 2. This set was used in our earlier study [6]. For consistency with that study, we do not reorder the matrix A. The problems come from a variety of application areas and are of different sizes and densities. As we expect that successfully using fp16 arithmetic will be most challenging for ill-conditioned problems, the problems were chosen because they all have a large estimated condition number (in the range $10^7 - 10^{16}$). Many are initially poorly scaled and some contain entries that overflow in fp16 and thus prescaling of A is essential. The right-hand side vector b is constructed by setting the solution x to be the vector of 1's. In Table 2, we also report the number of entries in the "scaled and squeezed" matrix A_l (see Line 2 of Algorithm 3). The squeezing discards all entries of the scaled matrix with absolute value less than $\tau_u = 10^{-5}$. We see that this can lead to the loss of a significant number of entries.

Our results are for the level-based incomplete Cholesky factorization preconditioner $IC(\ell)$ with $\ell = 2$ and 3 [33]. The number of entries in the incomplete factor L increases as the parameter ℓ increases. $IC(0)$ is a very simple preconditioner in which L has the same sparsity pattern as A. In practice, using very small ℓ may be sufficient for solving well-conditioned problems but, as shown in [6], the resulting preconditioner is often not powerful enough to successfully tackle ill-conditioned examples (particularly when computed using fp16 arithmetic). We refer to the $IC(\ell)$ factorizations computed using half and double precision arithmetic as fp16-$IC(\ell)$ and fp64-$IC(\ell)$, respectively. The key difference between the two versions is that for the former, during the incomplete factorization, we incorporate the safe checks for the scaling and update operations; for the fp64 version, tests for B1 breakdown are performed (B2 and B3 breakdowns were not encountered in our double precision experiments). The solves with L and L^T employ the L factor in double precision. This can be done by casting the data into double precision and making an explicit copy of L; this negates the important benefit that half precision offers of reducing memory requirements. Alternatively, the entries can be cast on the fly. This is straightforward to incorporate into a serial triangular solve routine, and only requires a temporary double precision array of length n. This is done in our software.

We use two precisions: $u_\ell = u_{16}$ for the incomplete factorization and $u_r = u_g = u_p = u = u_{64}$, where u_{16} and u_{64} denote the unit roundoffs in fp16 and fp64 arithmetic, respectively. That is, we aim to achieve double precision accuracy in the computed solution. The iterative refinement terminates when the normwise backward error for the computed solution satisfies

$$res = \frac{\|b - Ax\|_\infty}{\|A\|_\infty \|x\|_\infty + \|b\|_\infty} \le \delta = 10^3 \times u_{64}.$$

The implementation of GMRES is taken from the HSL software library [34] (`MI24` is a Fortran MGS-GMRES implementation), and its convergence tolerance is set to $u_{64}^{1/4}$ (see [6] for an explanation of this choice); for each application of GMRES (Step 5 of Algorithm 5) the limit on the number of iterations is 1000. Restarting is not used. In

Table 2 Statistics for our ill-conditioned test examples

Identifier	n	$nnz(A)$	$normA$	$normb$	$cond2$	$nnz(A_l)$
Boeing/msc01050	1050	1.51×10^{4}	2.58×10^{7}	1.90×10^{6}	4.58×10^{15}	4.63×10^{3}
HB/bcsstk11	1473	1.79×10^{4}	1.21×10^{10}	7.05×10^{8}	2.21×10^{8}	6.73×10^{3}
HB/bcsstk26	1922	1.61×10^{4}	1.68×10^{11}	8.99×10^{10}	1.66×10^{8}	6.59×10^{3}
HB/bcsstk24	3562	8.17×10^{4}	5.28×10^{14}	4.21×10^{13}	1.95×10^{11}	3.89×10^{4}
HB/bcsstk16	4884	1.48×10^{5}	4.12×10^{10}	9.22×10^{8}	4.94×10^{9}	5.24×10^{4}
Cylshell/s2rmt3m1	5489	1.13×10^{5}	9.84×10^{5}	1.73×10^{4}	2.50×10^{8}	5.09×10^{4}
Cylshell/s3rmt3m1	5489	1.13×10^{5}	1.01×10^{5}	1.73×10^{3}	2.48×10^{10}	5.07×10^{4}
Boeing/bcsstk38	8032	1.82×10^{5}	4.50×10^{11}	4.04×10^{11}	5.52×10^{16}	7.83×10^{4}
Boeing/msc10848	10848	6.20×10^{5}	4.58×10^{13}	6.19×10^{11}	9.97×10^{9}	3.02×10^{5}
Oberwolfach/t2dah_e	11445	9.38×10^{4}	2.20×10^{-5}	1.40×10^{-5}	7.23×10^{8}	4.88×10^{4}
Boeing/ct20stif	52329	1.38×10^{6}	8.99×10^{11}	8.87×10^{11}	1.18×10^{12}	6.30×10^{5}
DNVS/shipsec8	114919	3.38×10^{6}	7.31×10^{12}	4.15×10^{11}	2.40×10^{13}	7.70×10^{5}
GHS_psdef/hood	220542	5.49×10^{6}	2.23×10^{9}	1.51×10^{8}	5.35×10^{7}	2.66×10^{6}
Um/offshore	259789	2.25×10^{6}	1.44×10^{15}	1.16×10^{15}	4.26×10^{9}	1.17×10^{6}

$nnz(A)$ denotes the number of entries in the lower triangular part of A. $normA$ and $normb$ are the infinity norms of A and b. $cond2$ is a computed estimate of the condition number of A in the 2-norm. $nnz(A_l)$ is the number of entries in the lower triangular part of the matrix after scaling and squeezing

Table 3 Results for IC-GMRES-IR (Algorithm 5) using fp16-$IC(\ell)$ and fp64-$IC(\ell)$ preconditioners ($\ell = 2, 3$) with no look-ahead and with look-ahead (Section 3.2)

Identifier	No look-ahead its $(n1, n2)$	With look-ahead its $(n1)$	No look-ahead its $(n1, n2)$	With look-ahead its $(n1)$
	fp16-$IC(2)$		fp16-$IC(3)$	
Boeing/msc01050	65 (1, 0)	84 (4)	62 (1, 0)	81 (4)
HB/bcsstk24	428 (0, 1)	428 (1)	418 (0, 1)	418 (1)
Um/offshore	2013 (0, 4)	129 (5)	40 (0, 4)	40 (4)
	fp64-$IC(2)$		fp64-$IC(3)$	
Boeing/msc01050	24 (0, 0)	69 (4)	25 (0,0)	69 (4)
HB/bcsstk11	201 (0, 0)	174 (1)	29 (0,0)	29 (0)
Cylshell/s3rmt3m1	102 (0, 0)	102 (0)	NC (0, 0)	426 (1)
Boeing/ct20stif	NC (0, 0)	1940 (2)	1332 (0, 0)	1368 (1)
GHS_psdef/hood	‡ (0, 0)	568 (5)	‡ (0, 0)	407 (4)
Um/offshore	‡ (0, 0)	128 (5)	‡ (0, 0)	37 (4)

its is the total number of GMRES iterations and $(n1, n2)$ are the numbers of times B1 and B3 breakdowns are detected ($n2$ is nonzero only for fp16-$IC(\ell)$ with no look-ahead). NC indicates that on an inner iteration the requested GMRES accuracy of $u_{64}^{1/4}$ was not achieved within the limit of 1000 iterations. ‡ indicates failure to compute useful factors because of enormous growth in the entries

the tables of results, NC denotes that this limit has been exceeded without the GMRES convergence tolerance being achieved.

Our first experiment looks at the effects of incorporating look-ahead. Table 3 presents results with no look-ahead and with look-ahead. Here we only include the test problems for which look-ahead has an effect. We make a number of observations. Incorporating look-ahead can improve robustness, particularly when using fp64 arithmetic. Without look-ahead, in fp64 arithmetic there can be very large growth in the size of some entries in the factors and this goes undetected (no B1 breakdowns occur with $\tau_u = 10^{-20}$). With look-ahead, growth did not happen in our tests on ill-conditioned problems. In fp16 arithmetic, look-ahead can replace B3 breakdown by B1 breakdown (e.g., HB/bcsstk24). Even if there are no B3 breakdowns, look-ahead can lead to a larger number of B1 breakdowns being flagged (see problem Boeing/msc01050) and hence a larger number of restarts, a larger shift and, consequently, a higher GMRES iteration count. Note that the iteration counts for $IC(3)$ are not guaranteed to be smaller than for $IC(2)$ (although they generally are).

Table 4 presents results for IC-GMRES-IR using a fp16-$IC(2)$ preconditioner with look-ahead and the GMW(β) strategy for $\beta = 0.5$, 10, and 100. B3 breakdown only occurs for GMW(100) (there is a single B3 breakdown for examples HB/bcsstk24

Table 4 Results for IC-GMRES-IR (Algorithm 5) using a fp16-$IC(2)$ preconditioner with the GMW(β) strategy (Section 3.4) and with look-ahead (Section 3.2)

Identifier	GMW(0.5) its $(n1, nmod)$	GMW(10) its $(n1, nmod)$	GMW(100) its $(n1, nmod)$	With look-ahead its $(n1)$
Boeing/msc01050	96 (0, 60)	65 (1, 0)	65 (1, 0)	84 (4)
HB/bcsstk11	1092 (0, 476)	NC (0, 310)	205* (0, 0)	205 (1)
HB/bcsstk26	786 (0, 476)	111 (1, 0)	111 (1, 0)	87 (1)
HB/bcsstk24	1018 (0, 446)	NC (0, 428)	$428^\sharp$ (0, 0)	428 (1)
HB/bcsstk16	41 (0, 26)	23 (0, 0)	23 (0, 0)	23 (0)
Cylshell/s2rmt3m1	787 (0, 584)	155 (0, 0)	155 (0, 0)	155 (0)
Cylshell/s3rmt3m1	2017 (1, 710)	630 (2, 0)	630 (2, 0)	630 (2)
Boeing/bcsstk38	1335 (1, 914)	313 (1, 0)	$313^\sharp$ (0, 0)	313 (1)
Boeing/msc10848	684 (0, 591)	81 (0, 0)	81 (0, 0)	81 (0)
Oberwolfach/t2dah_e	11 (0, 6)	7 (0, 0)	7 (0, 0)	7 (0)
Boeing/ct20stif	2139 (0, 4827)	1900 (2, 0)	1900 (2, 0)	1900 (2)
DNVS/shipsec8	2569 (1, 1)	2390 (1, 0)	2390 (1, 0)	1492 (1)
GHS_psdef/hood	2459 (0, 25074)	581* (0, 0)	581 (5, 0)	581 (5)
Um/offshore	NC (0, 3846)	NC (0, 5838)	2013 (4, 2)	129 (5)

its is the number of GMRES iterations and $(n1, nmod)$ are the numbers of times B1 breakdown is detected and the number of local modifications made by the GMW strategy. $\sharp$ and $*$ indicate B3 and B4 breakdowns, respectively. NC indicates that on an inner iteration the requested GMRES accuracy of $u_{64}^{1/4}$ was not achieved within the limit of 1000 iterations

Table 5 Results for problem Boeing/bcsstk38

β	0.1	0.2	0.3	0.4	0.45	0.5	0.6	0.7	0.8	0.9	1.0
$nmod$	7404	5751	3695	1891	1327	914	621	409	138	8	0
its	3542	2634	2068	2037	1642	1335	481	390	321	313	313

IC-GMRES-IR is run using a fp16-$IC(2)$ preconditioner computed with the GMW(β) strategy for a range of values of β (Section 3.4). $nmod$ and its are the numbers of local modifications made by the GMW strategy and GMRES iterations, respectively

and Boeing/bcsstk38 and for these a global shift is used). B4 breakdown happens only for GMW(10) applied to GHS_psdef/hood (5 occurrences for this example) and GMW(100) applied to HB/bcsstk11 (happens once); again a global shift is used to avoid breakdown. We see that with $\beta = 0.5$, for some examples a large number of local modifications ($nmod$) are made. This leads to the preconditioner being of poorer quality compared to the $IC(2)$ preconditioner with look-ahead. For $\beta = 10$, local modifications are only needed for a few problems (HB/bcsstk11, HB/bcsstk24 and Um/offshore). In each case, the resulting preconditioner is not successful. For GMW(100), local modifications are only made for Um/offshore; for all other test examples, GMW(100) is equivalent to $IC(2)$ with no look-ahead.

The sensitivity of the GMW(β) approach to the choice of β is reported on in Table 5 for problem Boeing/bcsstk38. As β increases, the number of local modifications to diagonal entries ($nmod$) steadily decreases and so too does the GMRES iteration count (its). For this example, for each $\beta \geq 0.4$, B1 breakdown was detected once and a global shift $\alpha = 10^{-3}$ was then employed.

Finally, results for IC-GMRES-IR using a fp64-$IC(2)$ preconditioner are given in Table 6. As we would expect, the number of breakdowns and the iteration counts are often less than for the fp16-$IC(2)$ preconditioner. If $\beta = 0.5$, the number of local modifications when using fp64 arithmetic is very similar to the number when using fp16 and the iteration counts are also comparable. For larger β, fp64 can result in a higher quality preconditioner but, as earlier, without look-ahead the computed preconditioner can be ineffective.

All the reported results employed our explicit safe tests for breakdown. We have also run the fp16 arithmetic experiments with the B1 to B4 breakdown tests replaced by IEEE exception handling. As expected, because in fp16 arithmetic $\tau_u = 10^{-5}$ and $x_{min} = \mathcal{O}(10^{-5})$, this led to the same number of restarts and hence the same iteration counts. However, by only testing the exception flag at the end of each major step of the factorization, this approach did not distinguish between the different types of breakdown. For the experiments using fp64 arithmetic, IEEE exception handling did not detect any problems and consequently, for some examples, this led to growth in the factor entries (exactly as in the case of no look-ahead).

Table 6 Results for IC-GMRES-IR (Algorithm 5) using a fp64-$IC(2)$ preconditioner with the GMW(β) strategy (Section 3.4) and with look-ahead (Section 3.2)

Identifier	GMW(0.5) its ($nmod$)	GMW(10) its ($nmod$)	GMW(100) its ($n1, nmod$)	With look-ahead its ($n1$)
Boeing/msc01050	78 (60)	24 (0)	24 (4, 0)	24 (0)
HB/bcsstk11	1087 (476)	201 (0)	201 (0, 0)	232 (0)
HB/bcsstk26	775 (476)	79 (0)	79 (0, 0)	79 (0)
HB/bcsstk24	913 (409)	89 (0)	89 (0, 0)	89 (0)
HB/bcsstk16	41 (26)	22 (0)	22 (0, 0)	22 (0)
Cylshell/s2rmt3m1	792 (585)	146 (0)	146 (0, 0)	146 (0)
Cylshell/s3rmt3m1	2901 (710)	102 (0)	102 (0, 0)	102 (0)
Boeing/bcsstk38	1301 (943)	141 (0)	141 (0, 0)	141 (0)
Boeing/msc10848	790 (600)	68 (0)	68 (0, 0)	68 (0)
Oberwolfach/t2dah_e	14 (6)	6 (0)	6 (0, 0)	6 (0)
Boeing/ct20stif	2122 (4847)	2036 (40)	NC (0, 0)	1940 (2)
DNVS/shipsec8	701 (4658)	354 (0)	354 (0, 55)	354 (0)
GHS_psdef/hood	2480 (25054)	NC (2013)	NC (0, 11998)	568 (5)
Um/offshore	NC (4094)	NC (6327)	‡(4, 4730)	128 (5)

its is the number of GMRES iterations and ($n1, nmod$) are the numbers of times B1 breakdown is detected and the number of local modifications made by the GMW strategy. For GMW(0.5) and GMW(10), $n1 = 0$ for all examples so is omitted. NC indicates that on an inner iteration the requested GMRES accuracy of $u_{64}^{1/4}$ was not achieved within the limit of 1000 iterations. ‡ indicates failure to compute useful factors because of enormous growth in the entries

5 Concluding remarks

Following on from our earlier study [6], in this paper we have illustrated the potential for using half precision arithmetic to compute incomplete factorization preconditioners that can be used to obtain double precision accuracy in the solution of highly ill-conditioned symmetric positive definite linear systems. In fp16 arithmetic, the danger of breakdown during the factorization of a sparse matrix is imminent and we must employ strategies that force computational robustness. To avoid breakdown, we have looked at global strategies plus a local modification scheme based on the GMW approach that has been used for dense matrices within the field of optimization. This employs a parameter β. Choosing a small β prevents breakdown during the factorization (in both fp16 and fp64 arithmetic) and there is no need to employ a global shift. However, the penalty is of poorer quality than that which is obtained by employing a simple global shifting approach. Thus, our recommendations are to always prescale the problem, to use a global shift, and to incorporate look-ahead. In addition, when developing software using fp16, monitoring for breakdown must be built in to ensure robustness. If this is done, then using low precision to compute an effective preconditioner appears to be feasible.

Once a Fortran compiler that supports bfloat16 becomes available, it would be very interesting to compare its performance to that of fp16. bfloat16 has the same exponent size as fp32 (single precision). Consequently, converting from fp32 to bfloat16 is easy: the exponent is kept the same and the significand is rounded or truncated from 24 bits to 8; hence overflow and underflow are not possible in the conversion. The disadvantage of bfloat16 is its lesser precision: essentially 3 significant decimal digits versus 4 for fp16. Another possible future direction is to explore the effects of different preorderings of A on the number of breakdowns and the quality of the low precision factors. Fill-reducing orderings can result in later entries in the factor being updated by more entries from the previous columns. Intuitively, this may lead to more breakdowns.

When using higher precision arithmetic, the potential dangers within an incomplete factorization algorithm can be hidden. As our experiments have demonstrated, a standard IC factorization using fp64 arithmetic without look-ahead can lead to an ineffective preconditioner because of growth in the size of the entries in the factors. Without careful monitoring (which is not routinely done), this growth may be unobserved but when subsequently applying the preconditioner, the triangular solves can overflow, resulting in the computation aborting.

Finally, we reiterate that, although our focus has been on symmetric positive definite systems, breakdown and/or large growth in factor entries is also an issue for the incomplete factorization of general sparse matrices. Again, safe checks (or the use of IEEE exception handling) need to be built into the algorithms and their implementations to guarantee robustness.

Acknowledgements We would like to thank John Reid for discussions on exception handling in Fortran and for carefully reading and commenting on a draft of this manuscript. We are also grateful to two anonymous referees for their insightful feedback that has led to improvements in the paper.

Author Contributions Both authors contributed equally to the manuscript (the developed of ideas, numerical experiments, and preparation of the text).

Funding The authors received no external grant funding for this research.

Data Availability The test matrices used in this study are taken from https://sparse.tamu.edu/

Declarations

Ethical Approval and consent to participate There was no ethics approval required for this research.

Competing interests The authors declare no competing interests.

References

1. Higham, N.J., Mary, T.: Mixed precision algorithms in numerical linear algebra. Acta Numer. **31**, 347–414 (2022). https://doi.org/10.1017/S0962492922000022
2. Carson, E., Higham, N.J.: A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. SIAM J. Sci. Comput. **39**(6), 2834–2856 (2017). https://doi.org/10.1137/17M1122918
3. Amestoy, P., Buttari, A., Higham, N.J., L'Excellent, J.-Y., Mary, T., Vieublé, B.: Five-precision GMRES-based iterative refinement. SIAM J. Matrix Anal. Appl. **45**(1), 529–552 (2024). https://doi.org/10.1137/23M1549079
4. Carson, E., Higham, N.J.: Accelerating the solution of linear systems by iterative refinement in three precisions. SIAM J. Sci. Comput. **40**(2), 817–847 (2018). https://doi.org/10.1137/17M1140819
5. Higham, N.J., Pranesh, S.: Exploiting lower precision arithmetic in solving symmetric positive definite linear systems and least squares problems. SIAM J. Sci. Comput. **43**(1), 258–277 (2021). https://doi.org/10.1137/19M1298263
6. Scott, J.A., Tůma, M.: Avoiding breakdown in incomplete factorizations in low precision arithmetic. ACM Trans. Math. Softw. **50**(2), Article 9 (2024). https://doi.org/10.1145/3651155
7. Eijkhout, V.: On the existence problem of incomplete factorisation methods. Lapack Working Note No. 144, UT-CS-99-435, http://www.netlib.org/lapack/lawns/ (1999)
8. Higham, N.J., Pranesh, S.: Simulating low precision floating-point arithmetic. SIAM J. Sci. Comput. **41**(5), 585–602 (2019). https://doi.org/10.1137/19M1251308
9. Chan, T.F., Vorst, H.A.: Approximate and incomplete factorizations. In: Parallel Numerical Algorithms, ICASE/LaRC Interdisciplinary Series in Science and Engineering IV. Centenary Conference, D.E. Keyes, A. Sameh and V. Venkatakrishnan, Eds., pp. 167–202. Kluver Academic Publishers, Dordrecht (1997). https://doi.org/10.1007/978-94-011-5412-3_6
10. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn., p. 528. SIAM, Philadelphia, PA (2003). https://doi.org/10.1137/1.9780898718003
11. Il'in, Y.M.: Iterative Incomplete Factorization Methods. World Scientific, Singapore (1992). https://doi.org/10.1142/1677
12. Scott, J.A., Tůma, M.: The importance of structure in incomplete factorization preconditioners. BIT Numer. Math. **51**, 385–404 (2011). https://doi.org/10.1007/s10543-010-0299-8
13. Scott, J.A., Tůma, M.: Algorithms for Sparse Linear Systems. Nečas Center Series, p. 242. Birkhäuser/Springer, Cham, (2023). https://doi.org/10.1007/978-3-031-25820-6
14. Meijerink, J.A., van der Vorst, H.A.: An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. Math. Comput. **31**, 148–162 (1977)

15. Meurant, G.: Computer Solution of Large Linear Systems. Elsevier, Amsterdam – Lausanne – New York – Oxford – Shannon – Singapore – Tokyo (1999)
16. Oktay, E., Carson, E.: Multistage mixed precision iterative refinement. Numer. Linear Algebra Appl. **29**(4), 2434 (2022). https://doi.org/10.1002/nla.2434
17. Demmel, J.W., Li, X.: Faster numerical algorithms via exception handling. IEEE Trans. Comput. **43**(8), 983–992 (1994). https://doi.org/10.1109/12.295860
18. Lindstrom, P., Lloyd, S., Hittinger, J.: Universal coding of the reals: alternatives to IEEE floating point. In: Proceedings of the Conference for Next Generation Arithmetic, pp. 1–14 (2018). https://doi.org/10.1145/3190339.3190344
19. Higham, N.J., Pranesh, S., Zounon, M.: Squeezing a matrix into half precision, with an application to solving linear systems. SIAM J. Sci. Comput. **41**(4), 2536–2551 (2019). https://doi.org/10.1137/18M1229511
20. Lin, C.-J., Moré, J.J.: Incomplete Cholesky factorizations with limited memory. SIAM J. Sci. Comput. **21**(1), 24–45 (1999). https://doi.org/10.1137/S1064827597327334
21. Scott, J.A., Tůma, M.: `HSL_MI28`: an efficient and robust limited-memory incomplete Cholesky factorization code. ACM Trans. Math. Softw. **40**(4), Article 24 (2014). https://doi.org/10.1145/2617555
22. Manteuffel, T.A.: An incomplete factorization technique for positive definite linear systems. Math. Comput. **34**, 473–497 (1980). https://doi.org/10.2307/2006097
23. Gill, P.E., Murray, W.: Newton-type methods for unconstrained and linearly constrained optimization. Math. Program. **7**, 311–350 (1974). https://doi.org/10.1007/BF01585529
24. Gill, P.E., Murray, W., Wright, M.H.: Practical Optimization. SIAM, 2nd edition, Philadelphia (2019). https://doi.org/10.1137/1.9781611975604
25. Eskow, E., Schnabel, R.B.: Algorithm 695: Software for a new modified Cholesky factorization. ACM Trans. Math. Softw. **17**, 306–312 (1991). https://doi.org/10.1145/114697.116806
26. Conn, A.R., Gould, N.I.M., Toint, Ph.L.: Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. Math. Program. **73**, 73–110 (1996). https://doi.org/10.1007/BF02592099
27. Fang, H., O'Leary, D.P.: Modified Cholesky algorithms: a catalog with new approaches. Math. Program. **115**(2, Ser. A), 319–349 (2008). https://doi.org/10.1007/s10107-007-0177-6
28. Schnabel, R.B., Eskow, E.: A new modified Cholesky factorization. SIAM J. Sci. Comput. **11**, 424–445 (1990). https://doi.org/10.1137/0911064
29. Schnabel, R.B., Eskow, E.: A revised modified Cholesky factorization algorithm. SIAM J. Optim. **11**, 1135–1148 (1990). https://doi.org/10.1137/S105262349833266X
30. Greenbaum, A.: Estimating the attainable accuracy of recursively computed residual methods. SIAM J. Matrix Anal. Appl. **18**, 535–551 (1997). https://doi.org/10.1137/S0895479895284944
31. Carson, E., Higham, N.J., Pranesh, S.: Three-precision GMRES-based iterative refinement for least squares problems. SIAM J. Sci. Comput. **42**(6), 4063–4083 (2020). https://doi.org/10.1137/20M1316822
32. Carson, E., Khan, N.: Mixed precision iterative refinement with sparse approximate inverse preconditioning. SIAM J. Sci. Comput. **45**(3), 131–153 (2023). https://doi.org/10.1137/22M1487709
33. Watts, J.W.: A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation. Soc. Pet. Eng. J **21**, 345–353 (1981)
34. HSL. A collection of Fortran codes for large-scale scientific computation. https://www.hsl.rl.ac.uk. (Accessed 2024)