LAPACK Working Note 165 / UCB//CSD-04-1344

Error Bounds from Extra Precise Iterative Refinement*

James Demmel† Yozo Hida‡ W. Kahan§ Xiaoye S. Li¶ Soni Mukherjee‖
E. Jason Riedy**

February 8, 2005

Abstract

We present the design and testing of an algorithm for iterative refinement of the solution of linear equations, where the residual is computed with extra precision. This algorithm was originally proposed in the 1960s [6, 22] as a means to compute very accurate solutions to all but the most ill-conditioned linear systems of equations. However two obstacles have until now prevented its adoption in standard subroutine libraries like LAPACK: (1) There was no standard way to access the higher precision arithmetic needed to compute residuals, and (2) it was unclear how to compute a reliable error bound for the computed solution. The completion of the new BLAS Technical Forum Standard [5] has recently removed the first obstacle. To overcome the second obstacle, we show how a single application of iterative refinement can be used to compute an error bound in any norm at small cost, and use this to compute both an error bound in the usual infinity norm, and a componentwise relative error bound.

We report extensive test results on over 6.2 million matrices of dimension 5, 10, 100, and 1000. As long as a normwise (resp. componentwise) condition number computed by the algorithm is less than $1/\max\{10,\sqrt{n}\}\varepsilon_w$, the computed normwise (resp. componentwise) error bound is at most $2\max\{10,\sqrt{n}\}\cdot\varepsilon_w$, and indeed bounds the true error. Here, n is the matrix dimension and ε_w is single precision roundoff error. For worse conditioned problems, we get similarly small correct error bounds in over 89.4% of cases.

*LAPACK Working Note 165, Computer Science Division Technical Report UCB//CSD-04-1344, University of California, Berkeley, 94720. This research was supported in part by the NSF Cooperative Agreement No. ACI-9619020; NSF Grant Nos. ACI-9813362 and CCF-0444486; the DOE Grant Nos. DE-FG03-94ER25219, DE-FC03-98ER25351, and DE-FC02-01ER25478; and the National Science Foundation Graduate Research Fellowship. The authors wish to acknowledge the contribution from Intel Corporation, Hewlett-Packard Corporation, IBM Corporation, and the National Science Foundation grant EIA-0303575 in making hardware and software available for the CITRIS Cluster which was used in producing these research results.

†Computer Science Division and Mathematics Dept., University of California, Berkeley, CA 94720 (demmel@cs.berkeley.edu).

‡Computer Science Division, University of California, Berkeley, CA 94720 (yozo@cs.berkeley.edu).

§Computer Science Division and Mathematics Dept., University of California, Berkeley, CA 94720 (wkahan@cs.berkeley.edu).

¶Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (xsli@lbl.gov).

‖Computer Science Division and Mathematics Dept., University of California, Berkeley, CA 94720.

**Computer Science Division, University of California, Berkeley, CA 94720 (ejr@cs.berkeley.edu).

Contents

1 Introduction

Iterative refinement is a technique for improving the accuracy of the solution of a system of linear equations $Ax = b$. Given some basic solution method (such as Gaussian Elimination with Partial Pivoting – GEPP), the basic algorithm is as follows:

> Input: An $n \times n$ matrix A, and an $n \times 1$ vector b
> Output: A solution vector $x^{(i)}$ approximating x in $Ax = b$, and
> an error bound $\approx \|x^{(i)} - x\|_\infty / \|x\|_\infty$
> Solve $Ax^{(1)} = b$ using the basic solution method
> $i = 1$
> repeat
> Compute residual $r^{(i)} = Ax^{(i)} - b$
> Solve $A\, dx^{(i+1)} = r^{(i)}$ using the basic solution method
> Update $x^{(i+1)} = x^{(i)} - dx^{(i+1)}$
> $i = i + 1$
> until $x^{(i)}$ is "accurate enough"
> return $x^{(i)}$ and an error bound

Algorithm 1: Basic iterative refinement

(Note that $x^{(i)}$ is a vector, and we use the notation $x_j^{(i)}$ to mean the j-th component of $x^{(i)}$.)

This can be thought of as Newton's method applied to the linear system $f(x) = Ax - b$. In the absence of error, Newton's method should converge immediately on a linear system, but the presence of rounding error in the inner loop of the algorithm prevents this immediate convergence and makes the behavior and analysis interesting.

The behavior of the algorithm depends strongly on the accuracy with which the residual $r^{(i)}$ is computed. We use *working precision* ε_w to denote the precision with which all input variables are stored. The basic solution method is used to solve $Ax = b$ and $A\, dx = r$ in working precision. In our numerical experiments, working precision is IEEE754 single precision, *i.e.* $\varepsilon_w = 2^{-24}$. Classical analyses of Wilkinson [6] and Moler [22] show that if the residual is computed to about double the working precision then as long as the condition number of A is not too large (sufficiently less than $1/\varepsilon_w$) the solution $x^{(i)}$ will converge to roughly working precision; this is the starting point for the analysis of Section 2 below. A more recent analysis of Skeel [29] considers computing the residual to working precision, and shows how this can improve backward stability but not necessarily accuracy; this is the version of iterative refinement implemented in LAPACK version 3.0 (see Algorithm 5 below). See [13, Chap. 12] for an overview of these schemes.

Section 2 presents a detailed error analysis of Algorithm 1 above, tracking the effects of rounding error in each line. We first use this analysis to derive and justify a stopping criterion and a reliable bound for the *normwise relative error*

$$\frac{\|x^{(i)} - x\|_\infty}{\|x\|_\infty}. \tag{1}$$

Here and later $x = A^{-1}b$ denotes the exact solution, assuming A is not singular.

Second, we observe that the entire algorithm is *column scaling invariant*. More precisely, if we assume that (1) our basic solution scheme is GEPP without any Strassen-like implementation [31],

(2) that no over/underflow occurs, and (3) C is any diagonal matrix whose diagonal entries are powers of the floating point radix β ($\beta = 2$ in the case of IEEE754 floating point standard arithmetic [2]), then replacing the matrix A by $A_c \equiv AC$ results in *exactly* the same roundoff errors being committed by Algorithm 1. Said another way, all the floating point numbers appearing throughout the algorithm change only in their exponents bits (by scaling by particular diagonal entries of C), not in their fraction bits: The exact solution x_c of the scaled system $A_c x_c = b$ satisfies $x_c = C^{-1}x$ where $Ax = b$, and every intermediate approximation $x_c^{(i)} = C^{-1}x^{(i)}$.

This means that a single application of Algorithm 1 (producing a sequence of approximations $x^{(i)}$) can be thought of as implicitly producing the sequence $x_c^{(i)}$ for the scaled system $A_c x_c = b$. This will mean that at a modest extra cost, we will be able to modify Algorithm 1 to compute the stopping criterion and error bound for x_c for any diagonal scaling C. (The extra cost is $O(n)$ per iteration, whereas one iteration costs $O(n^2)$ if A is a dense matrix.) In other words we will be able to cheaply compute a bound on the *scaled relative error*

$$\frac{\|C^{-1}(x^{(i)} - x)\|_\infty}{\|C^{-1}x\|_\infty} \tag{2}$$

for any scaling C.

Of the many C one might choose, a natural one would be $C \approx \mathrm{diag}(x_j)$, so that each component $x_{c,j} \approx 1$. This means that the scaled relative error (2) measures the componentwise relative error in the solution. There are two conditions for this to work. First, no component of x can equal 0, since in this case no finite componentwise relative error bound exists (unless the component is computed exactly). Second, the algorithm must converge (since C, which is computed on-the-fly, will affect the stopping criterion too).

Section 2 describes and analyzes the precise stopping criterion and error bound for Algorithm 1. One outcome of this analysis are two condition numbers that predict the success of iterative refinement. Let n be the matrix dimension and ε_w be the working precision. Then if the normwise condition number $\kappa_{\mathrm{norm}}(A) < 1/\gamma\varepsilon_w$, where $\gamma = \max\{10, \sqrt{n}\}$, the error analysis predicts convergence to a small normwise error and error bound. Similarly, if the componentwise condition number $\kappa_{\mathrm{comp}}(A) < 1/\gamma\varepsilon_w$, the error analysis predicts convergence to small componentwise error and error bound. This is borne out by our numerical experiments described below.

Our ultimate algorithm, Algorithm 3, is described in Section 3. Algorithm 3 differs from Algorithm 1 in several important ways:

- Both normwise and componentwise error bounds are computed from a single iteration, as mentioned above.

- If consecutive increments $dx^{(i)}$ are not decreasing rapidly enough, the algorithm switches to representing $dx^{(i)}$ in *doubled working precision*, *i.e.* by a pair of working precision arrays representing (roughly) the leading and trailing bits of $dx^{(i)}$ as though it were in double precision. Iteration continues subject to the same progress monitoring scheme. This significantly improves accuracy on the most ill-conditioned problems.

- Iteration halts if consecutive iterates $x^{(i)}$ differ little enough (measured in both normwise and componentwise senses), or if consecutive increments $dx^{(i)}$ do not decrease fast enough (despite representing $dx^{(i)}$ in doubled working precision), or if the maximum iteration count is exceeded.

- If a computed error bound exceeds a threshold (currently $\sqrt{\varepsilon_w}$), then it is set to 1, indicating that the algorithm cannot produce a reliable error bound.

Extensive numerical tests on over two million 100×100 test matrices are reported in Section 6. (Similar results were obtained on two million 5×5 matrices, two million 10×10 matrices, and $2 \cdot 10^5$ 1000×1000 matrices.) These test cases include a variety of scalings, condition numbers, and ratios of maximum to minimum components of the solution; see Section 5 for details on how the test cases were generated.

We summarize the results of these numerical tests. First we consider the normwise error and error bound. For not-too-ill-conditioned problems, those where $\kappa_{\text{norm}}(A) < 1/\gamma\varepsilon_w$, Algorithm 3 *always* computed an error bound of at most $2\gamma\varepsilon_w$, which exceeded the true error. Since the algorithm computes $\kappa_{\text{norm}}(A)$, these cases are easily recognized.

For even more ill-conditioned problems, with normwise condition numbers κ_{norm} ranging up past ε_w^{-2}, Algorithm 3 still gets similarly small normwise error bounds and true errors in 96.4% of cases. Convergence failure was reported in 3.4% of cases, and of the remaining 0.2% of cases (only 1800 out of nearly 1.2 million) the ratio of error bound to true error was in the range $(0.1, 10)$ all but 32 times, and never outside $(.02, 282)$. Details are reported in Section 6.1.

Next we consider the componentwise error and error bound. For not-too-ill-conditioned problems, those where $\kappa_{\text{comp}}(A) < 1/\gamma\varepsilon_w$, Algorithm 3 *always* computed an error bound of at most $2\gamma\varepsilon_w$, which again exceeded the true error. The number of iterations required was at most 4, with a median of 2. Since the algorithm computes $\kappa_{\text{comp}}(A)$, these cases are easily recognized.

For even more ill-conditioned problems, with componentwise condition numbers κ_{comp} ranging up past ε_w^{-2}, Algorithm 3 still gets similarly small componentwise error bounds and true errors in 94% of cases. Convergence failure was reported in 2.9% of cases, and of the remaining 3.1% of cases (45100 out of over 1.4 million) the ratio of error bound to true error was in the range $(0.1, 10)$ all but 1900 times, and never outside $(.007, 541)$. The median number of iterations for these ill-conditioned cases was 4, with a maximum of 33. Details are reported in Section 6.2.

The rest of Section 6 compares Algorithm 3 to Wilkinson's original algorithm (Algorithm 4) and the single precision routine currently in LAPACK (Algorithm 5) (it is more accurate than either one), and explores the impact of various design parameters on the behavior of Algorithm 3. In particular, we can make Algorithm 3 more or less aggressive in trying to converge on difficult problems; the above data is for our recommended "cautious" settings of these parameters.

Our use of extended precision is confined to two routines for computing the residual $r^{(i)} = Ax^{(i)} - b$, one where all the variables are stored in working precision, and one where $x^{(i)}$ is stored as a pair of vectors each in working precision: $r^{(i)} = Ax^{(i)} + Ax_t^{(i)} - b$. The first operation $r^{(i)} = Ax^{(i)} - b$ is part of the recently completed new BLAS standard [5], for which portable implementations exist [17]. It is critical for the accuracy of the routine. The second operation $r^{(i)} = Ax^{(i)} + Ax_t^{(i)} - b$ was not part of the new BLAS standard because its importance was not recognized. Nevertheless, it is straightforward to implement in a portable way using the same techniques in [17]. The importance of this second operation is quantified in Section 6.4.1, where it is shown to significantly improve accuracy for the most ill-conditioned matrices.

The rest of this paper is organized as follows. Section 2 describes the error analysis of Algorithms 1 and 2 in detail, including their invariance under column scaling. Section 3 describes the ultimate algorithm, Algorithm 3, including the parameters for the stopping criterion and error bound, and how the bound for (2) is computed. Section 4 describes related work, including our variation of Wilkinson's original algorithm (Algorithm 4) and LAPACK's version of Skeel's iterative refinement, (Algorithm 5) which we have modified to compute a componentwise error bound; both Algorithms 4 and 5 are compared in numerical experiments to Algorithm 3. Section 5 describes the test configuration, including the extra precision BLAS, platforms tested on, how test matrices are generated, and how the true error is computed. Section 6 presents the results of extensive numerical tests, and uses them to justify the the details of Algorithm 3 not justified by the error analysis of Section 2. Section 7 gives rare examples for which Algorithm 3 can fail. Section 8 presents the Fortran 77 [14] interface to the proposed new routines to be included in LAPACK. Finally Section 9 draws conclusions and describes future work.

2 Error Analysis

Algorithm 1 contains the basic computational steps but lacks termination criteria, error estimates, or specifications of the accuracy to which each step is performed. Let the true forward error of iteration i be denoted by $e^{(i)} \stackrel{\text{def}}{=} x^{(i)} - x$. The analysis below shows that both $\|e^{(i)}\|_\infty$ and $\|dx^{(i)}\|_\infty$ decrease in nearly the same way as rough geometric sequences until refinement hits its limiting precision. Monitoring the rate of decrease of $\|dx^{(i)}\|_\infty$ lets us estimate how $\|e^{(i)}\|_\infty$ decreases and so provides both termination criteria and an error estimate. Moreover, scaled solutions follow a similar geometric progression, which we use in Section 2.3 to estimate the *componentwise* error. For related analyses of iterative refinement, see [10, 13].

Refinement's precision is limited by the precision of its intermediate computations and storage formats. Computing with extra precision extends the limit, but mixing precisions requires accounting for errors in both computation and storage. We use the notation of a machine epsilon ε to model both concepts. A floating-point datum stored or computed with precision ε has a base-β significand with $-\log_\beta \varepsilon$ digits. Note that a precision ε_r is considered greater than precision ε_w when $\varepsilon_r < \varepsilon_w$. This analysis ignores over- and underflow.

Our refinement algorithm uses three different precisions distinguished with subscripts: ε_w, ε_x, and ε_r. The input data A and b are assumed to be stored exactly in the working precision ε_w. We also assume that any factorization of A is carried out in precision ε_w with results stored in ε_w. In our numerical experiments, ε_w is IEEE754 single precision [2], so $\beta = 2$ and $\varepsilon_w = 2^{-24}$. The residual $r^{(i)}$ and step $dx^{(i)}$ are also stored to precision ε_w, but the solution $x^{(i)}$ is stored and updated to precision $\varepsilon_x \le \varepsilon_w$, where possibly $\varepsilon_x \le \varepsilon_w^2$ if necessary for componentwise convergence. The criteria for choosing ε_x is discussed in Section 2.4. Residuals are calculated to extra precision ε_r with $\varepsilon_r \ll \varepsilon_w$ (typically $\varepsilon_r \le \varepsilon_w^2$). For our single-precision experiments, the residual is calculated in double precision with $\varepsilon_r = 2^{-53}$ and additional exponent range. The computed $x^{(i)}$ is carried either in single ($\varepsilon_x = \varepsilon_w = 2^{-24}$) or in a doubled single precision ($\varepsilon_x = \varepsilon_w^2 = 2^{-48}$). We base our experiments on single precision to ease testing; see Section 5 for details. Section 3 describes how, why, and when $x^{(i)}$ is carried to a doubled single precision.

The computed results $r^{(i)}$, $dx^{(i+1)}$, and $x^{(i+1)}$ from iteration i of Algorithm 1 satisfy the expressions

$$r^{(i)} = Ax^{(i)} - b + \delta r^{(i)} \quad \text{where} \quad |\delta r^{(i)}| \le n\varepsilon_r(|A| \cdot |x^{(i)}| + |b|) + \varepsilon_w |r^{(i)}|; \tag{3}$$

$$dx^{(i+1)} = (A + \delta A^{(i+1)})^{-1} r^{(i)} \quad \text{where} \quad |\delta A^{(i+1)}| \le 3n\varepsilon_w |L| \cdot |U|; \text{ and} \tag{4}$$

$$x^{(i+1)} = x^{(i)} - dx^{(i+1)} + \delta x^{(i+1)} \quad \text{where} \quad |\delta x^{(i+1)}| \le \varepsilon_x |x^{(i+1)}|. \tag{5}$$

Absolute values of matrices and vectors are interpreted elementwise.

The rounding error terms, those prefixed with δ, cannot be computed directly. Bounds for these terms are derived in standard ways [10, 13]. The residual is first computed to precision ε_r and then stored into precision ε_w. The error in Equation (5) is the error from representing the updated solution vector x in precision ε_x. The bound on the backward error $\delta A^{(i)}$ comes from the standard analysis of Gaussian elimination, absorbing row permutations into L. The error in Equation (4) affects iterative refinement's convergence rate but not its limiting accuracy so long as the errors do not prevent convergence [10, 13].

The only restriction we place on the solution method is homogeneity with respect to column scaling. The scaling and componentwise analysis in Sections 2.2 and 2.3 assumes that multiplying a column of A by a power of the floating point radix β only multiplies the corresponding column of $\delta A^{(i)}$ by the same factor. This is true for any reasonable implementation of Gaussian elimination *except* when Strassen-like algorithms are used for internal matrix products. Thus, the scaling analysis in Section 2.2 does not hold when Strassen-based BLAS are used. We have not tested such methods, and we do not know how they behave.

2.1 Normwise Error Estimate

Combining Equations (3)-(5) provides very similar recurrences governing the error $e^{(j)}$ and step $dx^{(j)}$. These recurrences provide an estimate of the normwise error at each step. To get a recurrence for $e^{(j+1)}$, one substitutes equation (3) into equation (4), solves equation (4) for $dx^{(j+1)}$, substitutes into (5), and subtracts x from both sides to get

$$e^{(j+1)} = (I + A^{-1}\delta A^{(j+1)})^{-1}(A^{-1}\delta A^{(j+1)}) \cdot e^{(j)} - (A + \delta A^{(j+1)})^{-1} \cdot \delta r^{(j)} + \delta x^{(j+1)}, \tag{6}$$

One may similarly derive

$$\begin{aligned} dx^{(j+1)} =& (I + A^{-1}\delta A^{(j+1)})^{-1}(A^{-1}\delta A^{(j)}) \cdot dx^{(j)} \\ &- (A + \delta A^{(j+1)})^{-1} \cdot (\delta r^{(j)} - \delta r^{(j+1)}) + (I + A^{-1}\delta A^{(j+1)}) \cdot \delta x^{(j+1)}. \end{aligned} \tag{7}$$

Assume for the moment that extra precision renders the $\delta r^{(j)}$, $\delta r^{(j+1)}$, and $\delta x^{(j+1)}$ terms negligible, leaving $\delta A^{(j)}$ and $\delta A^{(j+1)}$ as the only sources of error. This is a good approximation until convergence occurs. Before convergence, then, the above equations simplify to

$$e^{(j+1)} = (I + A^{-1}\delta A^{(j)})^{-1}(A^{-1}\delta A^{(j)}) \cdot e^{(j)}$$

and

$$dx^{(j+1)} = (I + A^{-1}\delta A^{(j+1)})^{-1}(A^{-1}\delta A^{(j)}) \cdot dx^{(j)} \ . \tag{8}$$

Comparing these two equations we see that $dx^{(j+1)}$ and $e^{(j+1)}$ decrease by being multiplied by very similar matrices at each step.

Rewriting $e^{(j+1)}$ as

$$e^{(j+1)} = (A^{-1}\delta A^{(j)})(I + A^{-1}\delta A^{(j)})^{-1} \cdot e^{(j)}$$

and multiplying by $(I + A^{-1}\delta A^{(j+1)})^{-1}$ leads to the "pseudo-error" expression

$$[(I + A^{-1}\delta A^{(j+1)})^{-1} \cdot e^{(j+1)}] = (I + A^{-1}\delta A^{(j+1)})^{-1}(A^{-1}\delta A^{(j)}) \cdot [(I + A^{-1}\delta A^{(j)})^{-1} \cdot e^{(j)}] \ . \tag{9}$$

This "pseudo-error" $(I + A^{-1}\delta A^{(j)})^{-1} \cdot e^{(j)}$ as well as $dx^{(j)}$ both decrease by being multiplied by identical matrices at each step. The pseudo-error differs from the true error $e^{(j+1)}$ by multiplication to the left with a matrix close to the identity. We conclude that the error $e^{(j)}$ and the increment $dx^{(j)}$ decrease in nearly the same way as long as roundoff terms $\delta r^{(j+1)}$ and $\delta x^{(j+1)}$ are negligible, and that the decrease is roughly geometric, with $\|dx^{(j)}\|_\infty$ and $\|e^{(j)}\|_\infty$ decreasing by a factor of at most about $\|(I + A^{-1}\delta A^{(j+1)})^{-1}(A^{-1}\delta A^{(j)})\|_\infty$.

Continuing to ignore roundoff terms $\delta r^{(j+1)}$ and $\delta x^{(j+1)}$, and assuming that the algorithm converges, we see that

$$\begin{aligned} x &= x^{(1)} - \sum_{j=2}^{\infty} dx^{(j)} \\ &= x^{(i)} - \sum_{j=i+1}^{\infty} dx^{(j)}, \end{aligned}$$

so that

$$e^{(i)} = x^{(i)} - x = \sum_{j=i+1}^{\infty} dx^{(j)},$$

and

$$\|e^{(i)}\|_\infty = \|x^{(i)} - x\|_\infty = \| \sum_{j=i+1}^{\infty} dx^{(j)} \|_\infty \le \sum_{j=i+1}^{\infty} \|dx^{(j)}\|_\infty \ . \tag{10}$$

Since we are assuming that $\|dx^{(j)}\|_\infty$ decreases geometrically, we substitute the easily computed maximum ratio

$$\rho_{\max} \stackrel{\text{def}}{=} \max_{j \le i} \frac{\|dx^{(j+1)}\|_\infty}{\|dx^{(j)}\|_\infty}. \tag{11}$$

into equation (10) to estimate

$$\|e^{(i)}\|_\infty \le \sum_{j=i+1}^{\infty} \|dx^{(j)}\|_\infty \le \|dx^{(i+1)}\|_\infty \sum_{j=i+1}^{\infty} \rho_{\max}^{j-i-1} = \|dx^{(i+1)}\|_\infty / (1 - \rho_{\max}) \ . \tag{12}$$

To account for rounding errors that become more dominant near convergence, we make sure our final bound is at least $\gamma\varepsilon_w$ with $\gamma = \max\{10, \sqrt{n}\}$. The lower bound of 10 protects against condition number underestimates and makes the bounds attainable for small systems. Altogether, the final normwise error bound is

$$B_{\text{norm}} \stackrel{\text{def}}{=} \max\left\{ \frac{\|dx^{(i+1)}\|_\infty / \|x^{(i)}\|_\infty}{1 - \rho_{\max}}, \gamma\varepsilon_w \right\} \approx \frac{\|x^{(i)} - x\|_\infty}{\|x\|_\infty} \stackrel{\text{def}}{=} E_{\text{norm}} \ . \tag{13}$$

For empirical comparisons of various alternative error bounds and justification of our choice B_{norm}, see section 6.4.

2.2 Equilibration and Choice of Scaled Norms

Equilibration refers to replacing the input matrix A by $A_s = R \cdot A \cdot C$ before factorization, where R and C are diagonal scaling matrices. The LAPACK equilibration routine we use first scales each row by dividing its entries by the row's largest magnitude entry. Columns are scaled likewise, taking into account the row scaling. We have modified this LAPACK routine to scale only by powers of the radix, so scaling does not introduce rounding errors unless some entry over- or underflows. This scaling leaves A_s with approximately unit row and column infinity-norms.

Equilibration reduces the likelihood of subsequent over- and underflow. Equilibration can also reduce ill-conditioning that is a by-product of ill-scaling. When A suffers from some forms of ill-scaling, the equilibrated $\kappa_s \equiv \kappa_\infty(A_s) = \|A_s^{-1}\|_\infty \|A_s\|_\infty$ can be much smaller than $\kappa_\infty(A)$. Later we will use various scaled condition numbers to separate cases where our algorithm performs reliably from those with no guarantees.

Directly applying the previous analysis to refining $A_s y^{(i)} = b_s$, where $b_s = R \cdot b$, provides an estimate of $\|y - y^{(i)}\|_\infty = \|C^{-1}(x - x^{(i)})\|_\infty$ rather than $\|x - x^{(i)}\|_\infty$. Assuming the user wants an error estimate for $x^{(i)}$ and not $y^{(i)}$, we must modify our error bounds.

To unscale the norm, the refinement algorithm applies the column scaling and computes

$$\|dx^{(i+1)}\|_\infty = \|Cdy^{(i+1)}\|_\infty \quad \text{and} \quad \|x^{(i)}\|_\infty = \|Cy^{(i)}\|_\infty$$

without computing $dx^{(i+1)}$ and $x^{(i)}$ directly. Algorithm 2 uses an equilibrated factorization but evaluates the error estimate in the user's norm. The algorithm also considers the user's solution$x^{(i)}$ rather than the scaled solution $y^{(i)}$ in the un-specified termination criteria.

```
Input: An n × n matrix A, and an n × 1 vector b
Output: A solution vector x^(i) approximating x in Ax = b, and
           an error bound ≈ ‖x^(i) − x‖∞/‖x‖∞
Equilibrate the system: A_s = R · A · C, b_s = R · b
Solve A_s y^(1) = b_s using the basic solution method
i = 1, ρ_max = 0, ‖dx^(1)‖∞ = ∞
repeat
    Compute residual r^(i) = A_s y^(i) − b_s
    Solve A_s dy^(i) = r^(i) using the basic solution method
    Compute ‖dx^(i+1)‖∞ = ‖C dy^(i+1)‖∞
    ρ_max = max{ρ_max, ‖dx^(i+1)‖∞/‖dx^(i)‖∞}
    n^(i) = ‖dx^(i+1)‖∞/‖x^(i)‖∞ = ‖C dy^(i+1)‖∞/‖C y^(i)‖∞
    Update y^(i+1) = y^(i) − dy^(i+1)
    i = i + 1
until x^(i) = C y^(i) is "accurate enough"
return x^(i) = C y^(i)
    and the normwise relative error bound max{ 1/(1−ρ_max) · n^(i), max{10, √n} · ε_w }
```

Algorithm 2: Iterative refinement with equilibration

Can we expect convergence in the user's norm (*i.e.* decreasing $\|dx^{(i)}\|_\infty$) with scaling C? To illustrate possible limits on C we continue to ignore $\delta r^{(i)}$ and $\delta x^{(i+1)}$ terms and observe that after equilibration by C equation (8) becomes

$$dx^{(i+1)} = Cdy^{(i+1)} = C(I + A_s^{-1}\delta A_s)^{-1} A_s^{-1} \delta A_s C^{-1} dx^{(i)} \stackrel{\text{def}}{=} C\hat{A}_s C^{-1} \cdot dx^{(i)}$$

so

$$\|dx^{(i+1)}\|_\infty \le \|C\hat{A}_s C^{-1}\|_\infty \cdot \|dx^{(i)}\|_\infty \quad .$$

To guarantee that $\|dx^{(i+1)}\|_\infty$ decreases, we ask what limits on C guarantee that $\|C\hat{A}_s C^{-1}\|_\infty < 1$. This last inequality may be rewritten in the equivalent form $C|\hat{A}_s|C^{-1}\mathbf{1} < \mathbf{1}$, where $\mathbf{1}$ is the vector of all ones, or $|\hat{A}_s|c < c$ where $c = C^{-1}\mathbf{1}$ is the vector of diagonal entries of C^{-1}. This last inequality limits the entries of C^{-1} to a certain homogeneous polytope. The smaller $\hat{A}_s$ is, the more values of c this polytope includes.

But clearly if some entry of c is too small (some diagonal entry of C is too large), then there may be convergence difficulties, even if A_s is well-conditioned. Because row scaling does not change $\|x\|_\infty$ or $\|y\|_\infty$ (barring over- and underflow), our termination criteria are row-scaling invariant. To identify potentially difficult cases, we may use a row-scaled condition number

$$\kappa_{\text{norm}} = \kappa_\infty(A_s \cdot C^{-1}) = \kappa_\infty(R \cdot A). \tag{14}$$

Here R and C are the equilibration factors from our modified LAPACK scaling routine. Note that typically $\kappa_{\text{norm}} \le \kappa_\infty(A)$, and κ_{norm} may be much smaller if A is badly row equilibrated.

2.3 Componentwise Error Estimate

Just as we could estimate the infinity norm's error bound for $x^{(i)} = Cy^{(i)}$ in Algorithm 2, we estimate an error bound in the infinity norm for *any* diagonally scaled $z^{(i)} = C_z y^{(i)}$. For example, if we were able to choose C_z so that $C_z y = \mathbf{1}$, *i.e.* so $z^{(i)}$ is converging to the vector of all ones, then the infinity norm error in $z^{(i)}$ would be identical to the largest relative error in any component. We could then modify Algorithm 2 to keep track of different values of $n^{(i)}$ and $\rho_{\max}$ (call them $n_z^{(i)}$ and $\rho_{\max,\text{z}}$), and so different error bounds for both C and C_z.

The only tricky part is that we seemingly need to know the answer y in order to determine C_z. If some components of y change significantly in early iterations, then C_z will change significantly and our estimates for $\rho_{\max,\text{z}}$ and $n_z^{(i)}$ will be poor. In practice, however, we only need to know each component of y approximately, so our approach is to wait until the relative change in each component of y is at most 0.25, and then choose C_z so that $C_z y^{(i)} = \mathbf{1}$. Note that C_z is only used to compute error bounds, not the iterates themselves, so there is no need to round entries of C_z to the nearest powers of β as was the case with the equilibration matrix C. With this choice of C_z, our componentwise relative error bound becomes

$$B_{\text{comp}} \stackrel{\text{def}}{=} \max\left\{\frac{\|C_z dy^{(i+1)}\|_\infty}{1-\rho_{\max,\text{z}}}, \gamma\varepsilon_w\right\} \approx \max_k \left|\frac{x_k^{(i)} - x_k}{x_k}\right| \stackrel{\text{def}}{=} E_{\text{comp}}. \tag{15}$$

Here $\gamma = \max\{10, \sqrt{n}\}$ as in Section 2.1, protecting against condition number underestimates and making the limit attainable for small systems.

As discussed in the previous section, we expect convergence difficulties when diagonal entries of C_z are widely varying. Again we use a scaled condition number to determine potentially difficult cases. This "componentwise" condition number is

$$\kappa_{\text{comp}} = \kappa_\infty(A_s \cdot C_z^{-1}) = \kappa_\infty(A_s \cdot \text{diag}(y)) = \kappa_\infty(R \cdot A \cdot \text{diag}(x)). \tag{16}$$

The same technique may be applied to LAPACK's current single precision refinement algorithm in SGERFS. Scaling by $x^{(i)}$ in LAPACK's forward error estimator produces the loose componentwise error estimate

$$B_{\text{comp}} \approx \|C_z^{-1} \cdot |A^{-1}| \cdot (|r| + (n+1)\varepsilon_w(|A||y| + |b|))\|_\infty.$$

As we will see in Section 6, this estimate is reliable for single-precision refinement. However, LAPACK's refinement routine targets *backward* error, so the forward errors can be quite large for ill-conditioned linear systems.

2.4 Termination Criteria and Employing Additional Precision

The contributions of δr and δx determine the remaining piece of our algorithm: termination criteria. These errors prevent $\|e^{(i)}\|_\infty$ from becoming arbitrarily small, halting its geometric decrease. Equation (13)'s error estimate relies on that geometric decrease, so the error bound becomes unreliable once δr and δx become significant. To maintain a reliable error estimate, our algorithm employs three termination criteria. Refinement halts when

1. the error estimate stops decreasing.
2. the step $dx^{(i+1)}$ fails to change $x^{(i)}$ significantly, or
3. we have invested too much work (iterated too many times).

Additionally, we increase the precision used to store $x^{(i)}$ the first time the error estimate stops decreasing (if we have not already converged and terminated). This section deals with the normwise stopping criteria. The componentwise criteria are similar. Section 3 makes explicit the criteria and their interplay.

Refinement's failure to decrease the error estimate is strong empirical evidence that the errors δr and δx have become significant. We determine that refinement has reached its limiting precision when the step to the next solution no longer satisfies $\|dx^{(i+1)}\|_\infty < \rho_{\text{thresh}}\|dx^{(i)}\|_\infty$ for a threshold ρ_{thresh} satisfying $0 < \rho_{\text{thresh}} \leq 1$. Thus refinement will

$$\text{stop if} \quad \frac{\|dx^{(i+1)}\|_\infty}{\|dx^{(i)}\|_\infty} \geq \rho_{\text{thresh}}. \tag{17}$$

The experiments in Section 6 justify setting $\rho_{\text{thresh}} = 0.5$ as a *cautious* approach that rarely yields significant over- or underestimates of errors. Setting $\rho_{\text{thresh}} = 0.9$ enables an *aggressive* approach that converges more often on extremely ill-conditioned matrices. Additionally, we use this criterion to trigger changing ε_x from ε_w to $\approx \varepsilon_w^2$; see Section 3 for details.

Geometrically decreasing step sizes drive $dx^{(i+1)}$ down to where it no longer significantly changes $x^{(i)}$. Further refinement cannot improve the solution. Thus refinement will

$$\text{stop if} \quad \frac{\|dx^{(i+1)}\|_\infty}{\|x^{(i)}\|_\infty} \leq \varepsilon_w. \tag{18}$$

Continuing refinement beyond either of these criteria rarely changes the computed solution but may affect the error estimate. While criterion (17) is satisfied, updating a stored $\|dx^{(i+1)}\|_\infty/\|dxi\|_\infty$ leads to overestimating the true error. Updating a stored, final $\|dx^{(i+1)}\|_\infty/\|x^{(i)}\|_\infty$ while either criterion holds may severely underestimate the error. Each of these criteria can be applied to any scaled norm $\|C(\cdot)\|_\infty$ to check progress and conversion.

The final stopping criterion is a purely pragmatic limit on the number of iterations, which we call i_{thresh}:

$$\text{stop if} \quad i > i_{\text{thresh}}. \tag{19}$$

As seen in Section 6, an aggressively large ρ_{thresh} (*i.e.* close to 1) can require a large number of iterations to satisfy criteria (17)-(18). A low ρ_{thresh} can also be used to limit the maximum number of iterations, but we have found that a low ρ_{thresh} prematurely terminates some refinements that are converging acceptably quickly, except for a single step with a large ratio $\|dx^{(i+1)}\|_\infty / \|dx^{(i)}\|_\infty$.

3 Algorithmic Details

Using the error estimates and termination criteria established above, we now describe our ultimate iterative refinement procedure, Algorithm 3. Our ultimate goal is an implementation with small run-time relative to Gaussian elimination. All norms in this section are the infinity norm. The algorithm refers to an auxiliary vector y, but the implementation stores y in place of x to minimize storage. We first scale A by R and C to $A_s = R \cdot A \cdot C$, where R and C are diagonal equilibration matrices with each diagonal entry being a power of the radix β. Then we solve the equilibrated system $A_s y = b_s$, where $y = C^{-1}x$, $b_s = Rb$. We perform the triangular factorization on the equilibrated A_s and proceed to refine $A_s y = b_s$. We also call a condition number estimator to estimate $\kappa_{\text{s}} = \kappa_\infty(A_s)$ for later use.

Algorithm 3 tracks the convergence state of x and z in the variables x-state $\in$ {working, no-progress, converged} and z-state $\in$ {unstable, working, no-progress, converged}, respectively. State variable y-scheme $\in$ {single, double} denotes the precision used for storing and computing with y. The refinement loop exits either from a large iteration count or when both x and z are no longer in a working state (see line 18).

The algorithm computes a step $dy^{(i+1)}$ for each iteration i. The error estimates and termination criteria require $\|x^{(i)}\|$ and $\|dx^{(i+1)}\|$. We transform vectors on the fly with $\|x^{(i)}\| = \|Cy^{(i)}\|$ and $\|dx^{(i+1)}\| = \|Cdy^{(i+1)}\|$ and avoid storing x or dx. Because C's entries are powers of the radix, the transformation is exact.

We obtain componentwise estimates as suggested in Section 2.3. Vector dz denotes the componentwise step in the estimates and termination criteria. We define $dz_k^{(i+1)} = dy_k^{(i+1)} / y_k^{(i)}$. Taking the norm $\|dz^{(i+1)}\| = \max_k |dy_k^{(i+1)}| / |y_k^{(i)}|$ gives the largest componentwise change. The quantity $\|dz^{(i+1)}\|$ is computed without storing dz.

The stopping criteria are applied to both the infinity norm error and componentwise relative error. Procedure new-x-state (called from line 15 in Algorithm 3) tests criteria (17) and (18) against $\|x^{(i)}\|$ and $\|dx^{(i+1)}\|$ for the normwise error. The componentwise error is already a relative error, so the tests in Procedure new-z-state (called from line 16 in Algorithm 3) require only $\|dz^{(i+1)}\|$.

If a solution fails to make progress by one measure but continues in another, the algorithm tests the "halted" measure to see if progress resumes. This test is on the first line of Procedure new-x-state and the second line of Procedure new-z-state. Criterion (17) should halt refinement only when enough error has accumulated to render all of the remaining steps inaccurate. The resumption tests allow for unexpected round-off in any single step and greatly improve convergence results on ill-conditioned problems.

Early in the iteration, the computed $y^{(i)}$ may be somewhat far from y componentwise, and our scaling by $y^{(i)}$ for componentwise quantities may be unreliable. Thus componentwise error is considered only after each component of the solution y is stable. We consider y stable if no component's relative change is more than dz_{thresh}. The error is roughly approximated by $\|dz^{(i+1)}\|$, so requiring a conservative dz_{thresh} should require *some* componentwise accuracy. This dz_{thresh} is set to 0.25 in our code, requiring only that the first bit is stable. With too large of a dz_{thresh}, the ratio of two consecutive dz grows too large for criterion (17) and stops refinement too early. So before checking for criterion (17), Procedure new-z-state re-evaluates the solution's stability and possibly marks it as unstable. The value of 0.25 appears to work well for binary arithmetic, but another value may be necessary for other bases.

We always stop if the iteration count exceeds i_{thresh}. The count measures the number of solves $Ax = b$ or $Adx^{(i+1)} = r^{(i)}$ performed. Criterion (19) differs from the other two because it can terminate refinement even while the current $dy^{(i+1)}$ improves the solution. In this case, the algorithm adds the $dy^{(i+1)}$ but still uses it to estimate the error. This could lead to a significant overestimate of the true error.

In Section 2 we discussed using additional precision to store and compute with the current iterate. A doubled arithmetic [19] provides this extra precision. Doubled arithmetic represents an extra-precise number as the sum of a pair of separately stored floating-point values, say f and f_t. The "tail" f_t serves as a slight correction to f that would be lost to round-off if $f + f_t$ were actually computed in floating-point arithmetic. Thus an additional $n \times 1$ vector $y_t^{(i)}$ is needed to hold the tails of the components of $y^{(i)}$. The tail fits into workspace already available, and we can return $y^{(i)}$ as the final result without additional work.

Using a doubled precision $\varepsilon_x \leq \varepsilon_w^2$ for storing $y^{(i)}$ increases the cost of each iteration, penalizing common, "easy" linear systems. We only use this extra precision when it appears necessary. The progress criterion (17) triggers when the errors δr and δx become significant. The δr is already $O(\varepsilon_w^2)$, so the only action remaining is to reduce δx. So if either the normwise error or the componentwise error stops making progress before convergence, the algorithm doubles the representation used for $y^{(i)}$. Extra precision in $y^{(i)}$ often leads the algorithm to good componentwise error even when the true solution has components of widely different magnitudes; see section 6 for details.

In our earlier experiments, we found that a variant of Algorithm 3 without the test in line 14 could drastically underestimate the solution's componentwise errors even for apparently well-conditioned systems when measured by $\kappa_s = \kappa_\infty(A_s)$. As explained in Section 7.2, this is the wrong condition number for the componentwise result. The condition number $\kappa_{\text{comp}} = \kappa_\infty(R \cdot A \cdot \operatorname{diag} x) = \kappa_\infty(A_s \cdot \operatorname{diag} y)$ more accurately reflects the contributions of round-off in the update step. The test in line 14 uses $\kappa_\infty(A_s) \cdot \kappa_\infty(\operatorname{diag} y) \geq \kappa_{\text{comp}}$ to trigger using extra precision to store and update y. This eliminates the underestimates at little cost.

The initialization in line 4 plays three roles. First, it ensures that $\rho_{\max}$ computed on the first iteration is 0. Second, termination because $i_{\text{thresh}} = 1$ will return error bounds of ∞. Finally, termination with $\mathsf{z\text{-}state} = \mathsf{unstable}$ will return a componentwise error estimate of ∞. The tests in lines 23 and 24 store the final $\|dx\|$ or $\|dz\|$ if the iteration count exceeds i_{thresh} while x or z is still making progress.

The states and transitions are shown graphically in Figure 1 using UML 1.4 notation [23]. The transitions are guarded by expressions in square brackets ([ ]) or triggered by events. The only event is incr-prec, which is raised in internal transitions to trigger increased precision. Note that the algorithm can terminate without using extra precision for y. The precision is raised only when necessary.

Input: An $n \times n$ matrix A, an $n \times 1$ vector b
Output: A solution vector $x^{(i)}$ approximating x in $Ax = b$,
    a normwise error bound $\approx \|x^{(i)} - x\|/\|x\|$, and
    a componentwise error bound $\approx \max_k |x_k^{(i)} - x_k|/|x_k|$
Equilibrate the system: $A_s = R \cdot A \cdot C$, $b_s = R \cdot b$
Estimate $\kappa_\mathrm{s} = \kappa_\infty(A_s)$
Solve $A_s y^{(1)} = b_s$ using the basic solution method
$\|dx^{(1)}\| = \|dz^{(1)}\| = \mathsf{final\text{-}relnorm}_x = \mathsf{final\text{-}relnorm}_z = \infty$
$\rho_{\max,\mathrm{x}} = \rho_{\max,\mathrm{z}} = 0.0$, x-state = working, z-state = unstable, y-scheme = single
for $i = 1$ to i_thresh do
    // **Compute residual in precision ε_r**
    if y-scheme = single then $r^{(i)} = A_s y^{(i)} - b_s$
    else $r^{(i)} = A_s(y^{(i)} + y_t^{(i)}) - b_s$, using doubled arithmetic
    // **Compute correction to $y^{(i)}$**
    Solve $A_s\, dy^{(i+1)} = r^{(i)}$ using the basic solution method
    // **Check error-related stopping criteria**
    Compute $\|x^{(i)}\| = \|Cy^{(i)}\|$, $\|dx^{(i+1)}\| = \|Cdy^{(i+1)}\|$ and $\|dz^{(i+1)}\| = \max_j \left| dy_j^{(i+1)} / y_j^{(i)} \right|$
    if y-scheme = single and $\kappa_\mathrm{s} \cdot \max_j |y_j| / \min_j |y_j| \geq 1/\gamma\varepsilon_w$ then incr-prec = true
    Update x-state, $\rho_{\max,\mathrm{x}}$ with Procedure `new-x-state` below
    Update z-state, $\rho_{\max,\mathrm{z}}$ with Procedure `new-z-state` below
    // **Either update may signal** incr-prec **or may set its** final-relnorm **.**
    if x-state $\neq$ working and z-state $\neq$ working then **BREAK**
    if incr-prec then y-scheme = double, incr-prec = false, and $y_t^{(i)} = 0$
    // **Update solution**
    if y-scheme = single then $y^{(i+1)} = y^{(i)} - dy^{(i+1)}$
    else $(y^{(i+1)} + y_t^{(i+1)}) = (y^{(i)} + y_t^{(i)}) - dy^{(i+1)}$ in doubled arithmetic
if x-state = working then $\mathsf{final\text{-}relnorm}_x = \|dx^{(i+1)}\| / \|x^{(i)}\|$
if z-state = working then $\mathsf{final\text{-}relnorm}_z = \|dz^{(i+1)}\|$
return $x^{(i)} = Cy^{(i)}$,
    normwise error bound $\max\left\{ \frac{1}{1-\rho_{\max,\mathrm{x}}} \cdot \mathsf{final\text{-}relnorm}_x, \max\{10, \sqrt{n}\} \cdot \varepsilon_w \right\}$, and
    componentwise error bound $\max\left\{ \frac{1}{1-\rho_{\max,\mathrm{z}}} \cdot \mathsf{final\text{-}relnorm}_z, \max\{10, \sqrt{n}\} \cdot \varepsilon_w \right\}$

Algorithm 3: New iterative refinement

Input: Current x-state, $\|x^{(i)}\|$, $\|dx^{(i)}\|$, $\|dx^{(i-1)}\|$, y-scheme
Output: New x-state and $\rho_{\max,\mathrm{x}}$, possibly signaling incr-prec or updating $\mathsf{final\text{-}relnorm}_x$
if x-state = no-progress and $\|dx^{(i+1)}\| / \|dx^{(i)}\| \leq \rho_\mathrm{thresh}$ then x-state = working
if x-state = working then
    if $\|dx^{(i+1)}\| / \|x^{(i)}\| \leq \varepsilon_w$ then x-state = converged // **Criterion** (18)**, tiny** dx
    else if $\|dx^{(i+1)}\| / \|dx^{(i)}\| > \rho_\mathrm{thresh}$ then
        if y-scheme = single then incr-prec = true
        else x-state = no-progress // **Criterion** (17)**, lack of progress**
    else $\rho_{\max,\mathrm{x}} = \max\{\rho_{\max,\mathrm{x}}, \|dx^{(i+1)}\| / \|dx^{(i)}\|\}$
    if x-state $\neq$ working then $\mathsf{final\text{-}relnorm}_x = \|dx^{(i+1)}\| / \|x^{(i)}\|$

Procedure `new-x-state`

Input: Current z-state, $\|dz^{(i)}\|$, $\|dz^{(i-1)}\|$, y-scheme
Output: New z-state and $\rho_{\max,z}$, possibly signaling incr-prec or updating final-relnorm$_z$
if z-state = unstable and $\|dz^{(i+1)}\| \leq dz_{\text{thresh}}$ then z-state = working
if z-state = no-progress and $\|dz^{(i+1)}\|/\|dz^{(i)}\| \leq \rho_{\text{thresh}}$ then z-state = working
if z-state = working then
    if $\|dz^{(i+1)}\| \leq \varepsilon_w$ then z-state = converged // **Criterion (18), tiny** dz
    else if $\|dz^{(i+1)}\| > dz_{\text{thresh}}$ then
        z-state = unstable, final-relnorm$_z$ = ∞, $\rho_{\max,z} = 0.0$
    else if $\|dz^{(i+1)}\|/\|dz^{(i)}\| > \rho_{\text{thresh}}$ then
        if y-scheme = single then incr-prec = true
        else z-state = no-progress // **Criterion (17), lack of progress**
    else $\rho_{\max,z} = \max\{\rho_{\max,z}, \|dz^{(i+1)}\|/\|dz^{(i)}\|\}$
    if z-state $\neq$ working then final-relnorm$_z$ = $\|dz^{(i+1)}\|$

Procedure `new-z-state`

4 Related Work

Extra precise iterative refinement was proposed in the 1960s. In [6], Wilkinson *et al.* presents the Algol programs that perform the LU factorization, the triangular solutions, and the iterative refinement using $\varepsilon_r = \varepsilon_w^2$. The detailed algorithm is as follows (with some changes described afterward):

Input: An $n \times n$ matrix A, and an n-long vector b
Output: A solution vector $x^{(i)}$ approximating x in $Ax = b$,
    a normwise error bound $\approx \|x^{(i)} - x\|_\infty / \|x\|_\infty$
Solve $Ax^{(1)} = b$ using the basic solution method
$i = 1$
repeat
    Compute residual $r^{(i)} = Ax^{(i)} - b$
    Solve $A\,dx^{(i+1)} = r^{(i)}$ using the basic solution method
    Update $x^{(i+1)} = x^{(i)} - dx^{(i+1)}$
    $i = i + 1$
until $i > i_{\text{thresh}}$ or $\|dx^{(i)}\|_\infty / \|dx^{(i-1)}\|_\infty > 0.5$ or $\|dx^{(i)}\|_\infty / \|x^{(i)}\|_\infty \leq 2\varepsilon_w$
Compute $\rho_{\max,x} = \max_{1 \leq j \leq i} \|dx^{(j+1)}\|_\infty / \|dx^{(j)}\|_\infty$
return $x^{(i)}$ and a normwise error bound $B_{\text{norm}} = \max\left\{\frac{1}{1-\rho_{\max,x}} \frac{\|dx^{(i)}\|_\infty}{\|x^{(i)}\|_\infty},\ \sqrt{n}\varepsilon_w\right\}$

Algorithm 4: Wilkinson's iterative refinement

In the basic solution method, Wilkinson uses the Crout algorithm for LU factorization, in which the inner products are performed in extra precision. The Crout algorithm was natural in those pre-BLAS days, but it cannot use high level BLAS routines, and is slow on current hierarchical memory systems. As noted in Section 2, higher precision inner products in the LU factorization only change the constant in the $O(\cdot)$ term of Equation (4), and do not affect our overall error analysis. Therefore, Section 6's comparison uses the same optimized version of GEPP (with Level 3 BLAS) as the basic solution method for all flavors of iterative refinement since that is the natural thing to do with today's technology.

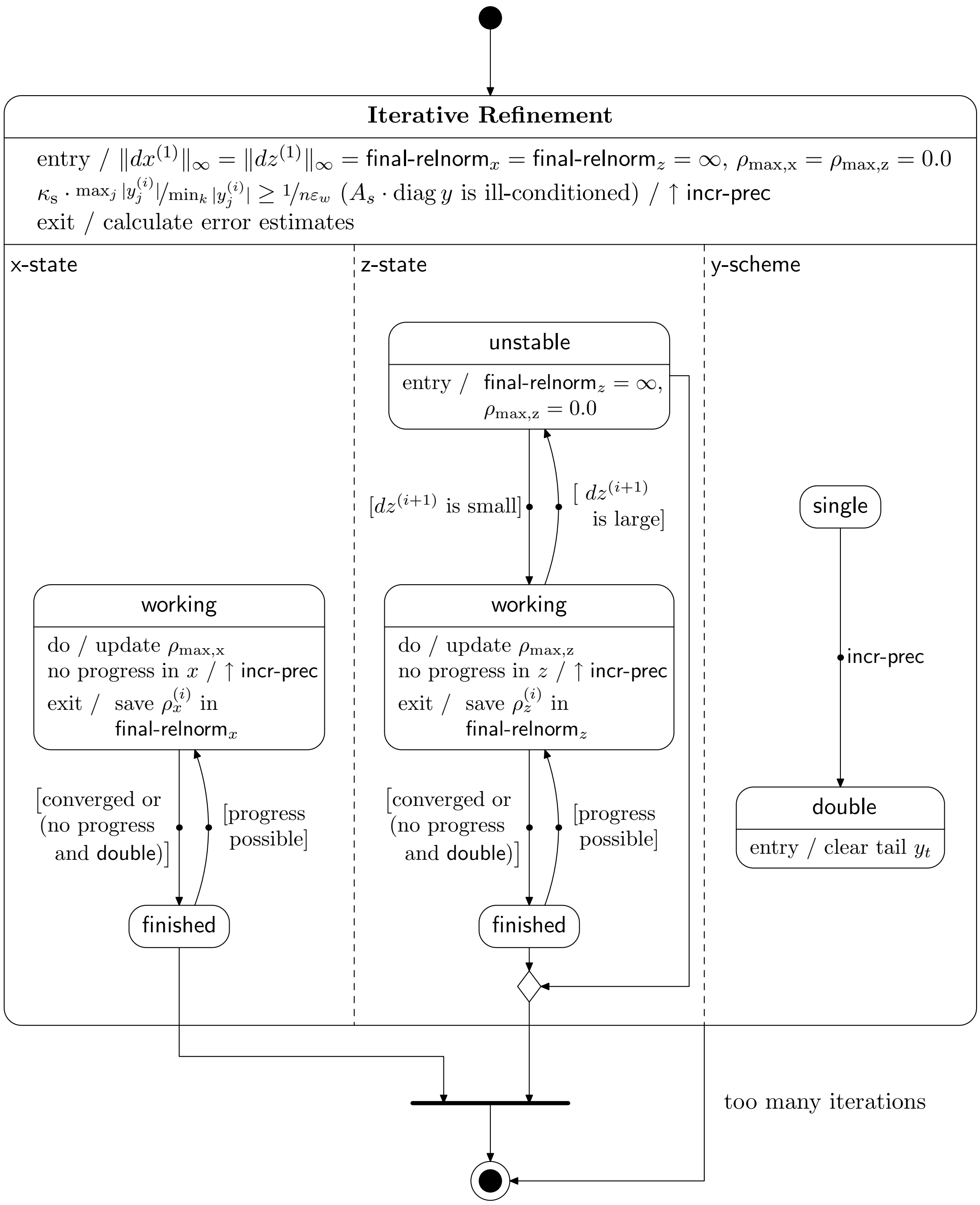


Figure 1: Overall statechart for Algorithm 3. The no-progress and converged states are merged into finished here.

Wilkinson's algorithm differs from ours in several ways:

- There is no initial equilibration in Wilkinson's algorithm.
- ρ_{thresh} is fixed to 0.5 in Wilkinson's algorithm.
- Wilkinson's algorithm does not store x to additional precision.
- Wilkinson's algorithm does not attempt to achieve componentwise accuracy.
- The original paper's algorithm [6] does not return an error bound. We add the error bound in Algorithm 4 based on our error analysis in Section 2.

There is no error analysis in [6]. But in [35], Wilkinson analyzes the convergence of the refinement procedure in the presence of roundoff errors from a certain type of scaled fixed point arithmetic. Moler extends Wilkinson's analysis to floating point arithmetic. Moler accounts for the rounding errors in the refinement process when the working precision is ε_w and the residual computation is in precision ε_r, and derives the following error bound [22, Eq. (11)]:

$$\frac{\|x^{(i)} - x\|_\infty}{\|x\|_\infty} \le [\sigma\kappa_\infty(A)\varepsilon_w]^i + \mu_1\varepsilon_w + \mu_2 n\kappa_\infty(A)\varepsilon_r \ ,$$

where σ, μ_1, and μ_2 are functions of the problem's dimension and condition number as well as refinement's precisions. Moler comments that "[if] A is not too badly conditioned" (meaning that $0 < \sigma\kappa_\infty(A)\varepsilon_w < 1$), the convergence will be dominated by the last two terms, and μ_1 and μ_2 are usually small. Furthermore, when ε_r is much smaller than ε_w (e.g., $\varepsilon_r \le \varepsilon_w^2$), the limiting accuracy is dominated by the second term. When $\varepsilon_r \le \varepsilon_w^2$, the stopping criterion he uses is $\|x^{(i)} - x^{(i-1)}\|_\infty \le \varepsilon_w\|x^{(1)}\|_\infty$. As for i_{thresh}, he suggests using the value near $-\log_{10}\varepsilon_r \approx 16$.

The use of higher precision in computing x was first presented as an exercise in Stewart's book [30, p. 207-208]. Stewart suggests that if x is accumulated in higher and higher precision, say in $\varepsilon_w, \varepsilon_w^2, \varepsilon_w^3, \ldots$ precisions, the residual will get progressively smaller. Eventually the iteration will give a solution with any desired accuracy. Kiełbasiński proposes an algorithm called binary cascade iterative refinement [16]. In this algorithm, GEPP and the first triangular solve for $x^{(0)}$ are performed in a base precision. Then during iterative refinement, both $r^{(i)}$ and $x^{(i+1)}$ are computed in increasing precision. Furthermore, the correction $dx^{(i)}$ is also computed in increasing precision by using the same increasing-precision iterative refinement process. That is probably why it has "cascade" in its name; the algorithm was in fact formulated recursively. Kiełbasiński analyzes the algorithm and shows that with a prescribed accuracy for x, you can choose a maximum precision required to stop the iteration. This algorithm requires arbitrary precision arithmetic, often implemented in software and considered too slow for wide use. We are not aware of any computer program that implements this algorithm.

A very different approach towards guaranteeing accuracy of a solution is to use interval arithmetic techniques [27, 28]. Interval techniques provide guaranteed bounds on a solution's error. However, intervals alone do not provide a more accurate solution. Intervals indicate when a solution needs improving and could guide application of extra precision. We will not consider interval algorithms further, although they are an interesting approach. We do not have a portable and efficient interval BLAS implementation and so cannot fairly compare our estimates with an interval-based algorithm.

Björck [3] nicely surveys the iterative refinement for linear systems and least-squares problems, including error estimates using working precision or extra precision in residual computation. Higham's book [13] gives a detailed summary of various iterative refinement schemes which have appeared through history. Higham also provides estimates of the limiting normwise and componentwise error. The estimates are not intended for computation but rather to provide intuition on iterative refinement's behavior. The estimates involve quantities like $\||A^{-1}| \cdot |A| \cdot |x|\|_\infty$ We experimented with approximating these error estimates using norm estimators. The additional level of estimation provided very inaccurate normwise and componentwise bounds.

Until now, extra precise iterative refinement was not adopted in standard libraries, such as LINPACK [11] and later LAPACK [1], mainly because there was no portable way to implement extra precision when the working precision was already the highest precision supported by the compiler. Therefore, the current LAPACK expert driver routines xGESVX only provide the working precision iterative refinement routines ($\varepsilon_r = \varepsilon_w$). Since iterative refinement can always ensure backward stability, even in working precision [13, Theorem 12.3], the LAPACK refinement routines use the componentwise backward error in the stopping criteria. The detailed algorithm is as follows, augmented as described at the end of Section 2.3 to compute a componentwise error bound, for later comparison with Algorithm 3:

Input: An $n \times n$ matrix A, and an n-long vector b
Output: A solution vector $x^{(i)}$ approximating x in $Ax = b$,
a normwise error bound $\approx \|x^{(i)} - x\|_\infty / \|x\|_\infty$, and
a componentwise error bound $\approx \max_k |x_k^{(i)} - x_k| / |x_k|$
Equilibrate the system: $A_s = R \cdot A \cdot C$, $b_s = R \cdot b$
Solve $A_s y^{(1)} = b_s$ using the basic solution method
while true do
    Compute residual $r^{(i)} = A_s y^{(i)} - b_s$ in working precision
    Compute backward error

$$\text{BERR}^{(i)} = \max_{1 \le k < n} \frac{|r_k^{(i)}|}{\left(|A_s||y^{(i)}| + |b_s|\right)_k}$$

    if $\text{BERR}^{(i)}/\text{BERR}^{(i-1)} > 0.5$ or $\text{BERR}^{(i)} \le \varepsilon_w$ or $i > i_{\text{thresh}}$ then BREAK
    Solve $A_s\, dy^{(i+1)} = r^{(i)}$ using the basic solution method
    Update $y^{(i+1)} = y^{(i)} - dy^{(i+1)}$
    $i = i + 1$
return $x^{(i)} = Cy^{(i)}$,
norm. error bound $B_{\text{norm}} = \text{FERR}^{(i)} \approx \frac{\||A_s^{-1}|\left(|r^{(i)}| + (n+1)\varepsilon_w(|A_s| \cdot |y^{(i)}| + |b_s|)\right)\|_\infty}{\|Cy^{(i)}\|_\infty}$,
comp. error bound $B_{\text{comp}} \approx \|C_{y^{(i)}}^{-1} \cdot |A_s^{-1}| \cdot \left(|r^{(i)}| + (n+1)\varepsilon_w(|A_s| \cdot |y^{(i)}| + |b_s|)\right)\|_\infty$.

Algorithm 5: LAPACK working precision iterative refinement with new componentwise error estimate

5 Testing Configuration

5.1 Review of the XBLAS

The XBLAS library is a set of routines for dense and banded BLAS routines, along with their extended and mixed precision versions; see Chapters 2 and 4 of the BLAS Technical Forum Standard [5]. Many routines in the XBLAS library allow higher internal precision to be used, enabling us to compute more accurate residuals. For example, general matrix-vector multiply `BLAS_sgemv_x` performs the matrix-vector multiplication $y \leftarrow \alpha Ax + \beta y$ in single, double, indigenous, or extra precision. For our experiments, A, x, y, α and β are input in precision $\varepsilon_r = 2^{-24}$, and internally double precision $\varepsilon_r = 2^{-53}$ is used. See [17] for further details.

In addition to the extra precision routines provided by the XBLAS, the doubled-x scheme in Algorithm 3 requires a new routine which we call `gemv2`. This routine takes a matrix A, three vectors x, y, and z, and three scalars α, β, and γ to compute

$$z \leftarrow \alpha Ax + \beta Ay + \gamma z \tag{20}$$

where the right hand side is evaluated with precision ε_r. This routine enables us to compute an accurate residual when x is kept in two words, x_{leading} and x_{trailing}:

$$r = b - A(x_{\text{leading}} + x_{\text{trailing}}). \tag{21}$$

5.2 Test Matrix Generation

To thoroughly test our algorithms, we need a large number of test cases with wide range of properties, including

- wide range of condition numbers,
- various distribution of singular values,
- well-scaled and ill-scaled matrices,
- matrices with first k columns nearly linearly dependent, and
- wide range of solution component sizes.

To achieve this goal, we generate test cases as follows:

1. Randomly pick a condition number κ with $\log_2 \kappa$ distributed uniformly in $[0, 26]$. This will be the (2-norm) condition number of the matrix before any scaling is applied.

2. We pick a set of singular values σ_i's from one of the following choices:

   (a) One large singular value: $\sigma_1 = 1$, $\sigma_2 = \cdots = \sigma_n = \kappa^{-1}$.

   (b) One small singular value: $\sigma_1 = \sigma_2 = \cdots = \sigma_{n-1} = 1$, $\sigma_n = \kappa^{-1}$.

   (c) Geometrical distribution: $\sigma_i = \kappa^{-\frac{i-1}{n-1}}$ for $i = 1, 2, \ldots, n$.

   (d) Arithmetic distribution: $\sigma_i = 1 - \frac{i-1}{n-1}(1 - \kappa^{-1})$ for $i = 1, 2, \ldots, n$.

3. Pick $k \in [1, n]$ (we randomly choose $k = 3$, $n/2$, and n). Move the largest and the smallest singular values (picked in step above) into the first k values so that leading k columns will be more ill-conditioned. Let Σ be the diagonal matrix with σ_i's on the diagonal. Form the matrix $\tilde{A}$ as follows:
$$\tilde{A} = U\Sigma \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} \tag{22}$$
where U, V_1, and V_2 are random orthogonal matrix with dimensions n, k, and $n-k$, respectively. If κ is large, this makes the first k leading columns of $\tilde{A}$ nearly singular, so that LU factorization of $\tilde{A}$ will encounter a small pivot at the k-th step. Orthogonal matrices are implicitly applied by performing random reflections from left and right.

4. To generate $\tilde{x}$, we first pick a number τ with $(\log_2 \tau)^{1/2}$ uniformly distributed in $[0, \sqrt{24}]$.* This will be the ratio between the largest and the smallest element (in magnitude):
$$\tau = \frac{\max_i |\tilde{x}_i|}{\min_i |\tilde{x}_i|}.$$
We generate $\tilde{x}$ by randomly choosing one of the following distributions:
   (a) One large component: $\tilde{x}_1 = 1$, $\tilde{x}_2 = \cdots = \tilde{x}_n = \tau^{-1}$.
   (b) One small component: $\tilde{x}_1 = \tilde{x}_2 = \cdots = \tilde{x}_{n-1} = 1$, $\tilde{x}_n = \tau^{-1}$.
   (c) Geometrical distribution: $\tilde{x}_i = \tau^{-\frac{i-1}{n-1}}$ for $i = 1, 2, \ldots, n$.
   (d) Arithmetic distribution: $\tilde{x}_i = 1 - \frac{i-1}{n-1}(1 - \tau^{-1})$ for $i = 1, 2, \ldots, n$.
   (e) Random: $\tilde{x}_i$'s are randomly chosen from the range $[\tau^{-1}, 1]$ such that $\log \tilde{x}_i$ is uniformly distributed.

   Note that the first four distributions are the same as the ones used in step (2). Finally, if one of the first four distribution is chosen, we multiply $\tilde{x}$ by a random number uniformly chosen from the range $[0.5, 1.5]$ to make the largest element not equal to 1 (so that all components of $\tilde{x}$ have full significand).

5. We then randomly column scale the matrix $\tilde{A}$ generated in step 3. We pick a scaling factor δ such that $(-\log_2 \delta)^{1/2}$ is uniformly distributed in $[0, \sqrt{24}]$.† Select two columns of $\tilde{A}$ at random and multiply by δ to produce the final input matrix A (rounded to single).

6. We compute $b = A\tilde{x}$ using double-double precision (using the XBLAS routine `BLAS_sgemm_x`), but rounded to single at the end.

7. Compute $x = A^{-1}b$ by using double precision GEPP with double-double precision iterative refinement. This corresponds to Algorithm 3 with IEEE754 double precision as working precision ($\varepsilon_w = 2^{-53}$) and double-double as residual precision ($\varepsilon_r \approx 2^{-105}$). Note that the "true" solution x thus obtained is usually not the same as $\tilde{x}$ that we started, due to rounding errors committed in step 6. This difference can be quite large if A is ill-conditioned.

*Thus $\log_2 \tau \in [0, 24]$, but the distribution is skewed to the left. We chose this distribution as to not overload our test samples with "hard" problems (in componentwise sense) with large $\kappa_{\text{comp}} = \kappa(RA\,\text{diag}(x)) = \kappa(A_s\,\text{diag}(y))$.

†As in step 4, we chose this distribution as to not overload our test samples with "hard" problems (in normwise sense) with large $\kappa(RA) = \kappa(A_s C^{-1})$.

Using the above procedure, two million 100×100, two million 10×10, and 2×10^5 1000×1000 test matrices were generated. Statistics for the 100×100 matrices will be discussed in the next section.

5.3 Test Matrix Statistics

In this section we present various statistics on the two million 100×100 test matrices generated. Similar statistics were obtained for the two million 5×5, two million 10×10, and 2×10^5 1000×1000 test cases as well. The six plots in figure 2 are histograms of various condition numbers that can be defined. The vertical blue line in each plot in figure 2 indicate where $\kappa = {}^1\!/\!\gamma\varepsilon_w$. For these systems, $\gamma = \max\{10, \sqrt{n}\} = 10$. Near the top of the vertical blue line, the two percentages indicates the fraction of total found on each side. All condition numbers that appear are ∞-norm condition numbers*. All condition numbers were computed in double precision, using a modification of `dgecon` condition number estimator. Hence even condition numbers approaching double precision limit are known accurately (up to the limits of the `rcond` estimator [13]).

Figure 2(a) shows the histogram of $\kappa_{\mathrm{s}} = \kappa_\infty(RAC) = \kappa_\infty(A_s)$, the condition number of the equilibrated system. This is a measure of the inherent difficulty of the system (measured in an appropriate norm). This condition number varies from about from about 57 to 1.6×10^{13}, with all but 2169 (0.11%) smaller than 10^{10}.

Figure 2(b) shows the histogram of $\kappa_{\mathrm{c}} = \kappa_\infty(C) = \|C\|_\infty$, the maximum column scaling factor computed during equilibration. This varies from 1 to $2^{35} \approx 3.4 \times 10^{10}$. Since the equilibration is only done if $\|C\|$ would be greater than 10, $C = I$ happens relatively often, appearing as a spike at $\kappa(C) = 1$ in the figure.

Figure 2(c) shows the histogram of $\kappa_{\mathrm{x}} = \kappa_\infty(\mathrm{diag}(x)) = {}^{\max_i |x_i|}\!/\!_{\min_i |x_i|}$, the ratio between the largest and smallest element (in magnitude) of x. This is a measure of how wildly the components of x varies. We see that κ_{x} varies from 1 to about 6.8×10^{13}, with all but 750 (0.04%) of them less than 10^{10}.

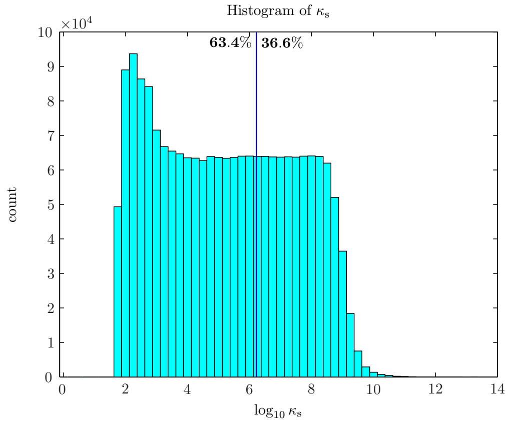
Figure 2(d) shows the histogram of $\kappa_{\mathrm{y}} = \kappa_\infty(\mathrm{diag}(y)) = {}^{\max_i |y_i|}\!/\!_{\min_i |y_i|}$, the ratio between the largest and smallest element (in magnitude) of y. This varies from 1 to approximately 8.5×10^{12}, with all but 1266 (0.06%) less than 10^{10}. This is a measure of how wildly the components of y varies, which gives some idea of the difficulty of getting componentwise accurate solution. The term κ_{y} appears naturally in the condition number for componentwise problem, since $\kappa_{\mathrm{comp}} = \kappa_\infty(A_s \,\mathrm{diag}(y)) \le \kappa_{\mathrm{s}}\kappa_{\mathrm{y}}$.

Figure 2(e) shows the histogram of κ_{norm}, varying from about 57 to 7.6×10^{17}. Recall that this is the condition number for the normwise problem: $\kappa_{\mathrm{norm}} = \kappa(RA) = \kappa(A_s C^{-1}) \le \kappa_{\mathrm{s}}\kappa_{\mathrm{c}}$. The vertical blue line divides the test matrices into two categories, according to κ_{norm}:

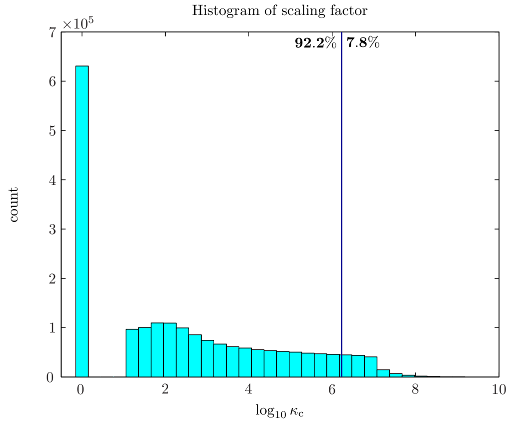
- **Normwise well-conditioned problems**. These are matrices with $\kappa_{\mathrm{norm}} \le {}^1\!/\!\gamma\varepsilon_w$, and perhaps more accurately described as "not too ill-conditioned" matrices. These are matrices where we hope to have an accurate solution in normwise sense. Most problems can be expected to fall in this category in practice. Of the two million test matrices, 821097 cases fall into this category.

- **Normwise ill-conditioned problems**. These are the rest of the matrices with $\kappa_{\mathrm{norm}} >$

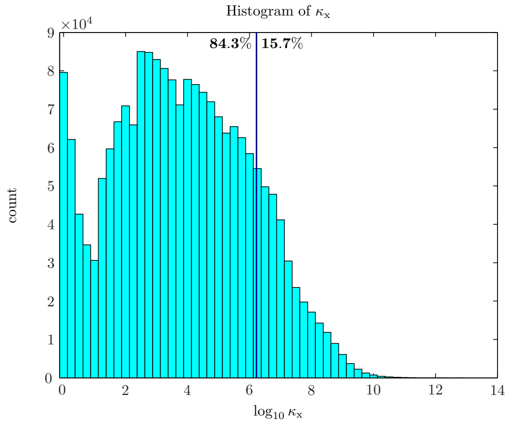
*LAPACK routines were modified to return κ_∞ rather than κ_1.



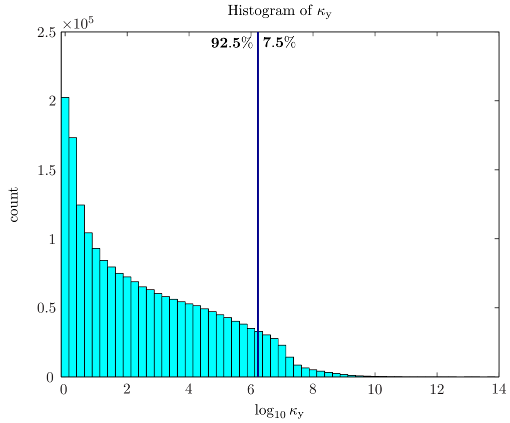
(a) $\kappa_s = \kappa(RAC)$



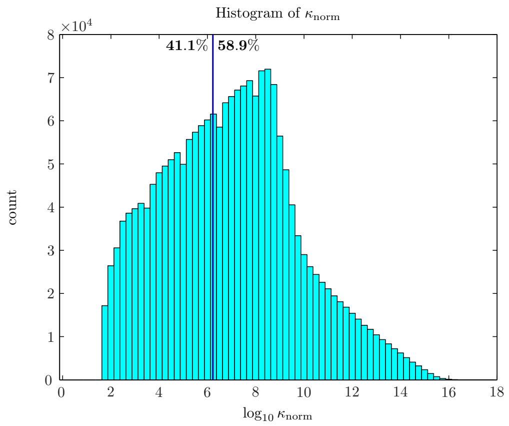
(b) $\kappa_c = \|C\|_\infty$



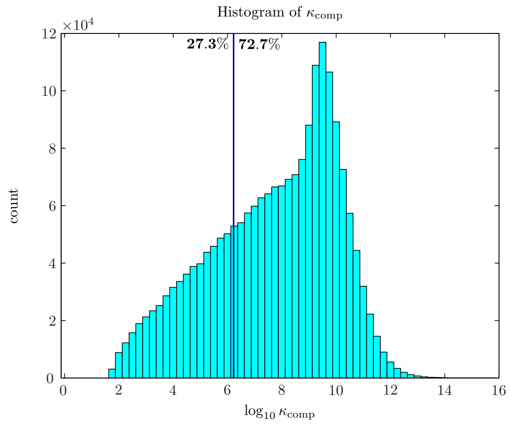
(c) $\kappa_x = {\max_i |x_i|}/{\min_i |x_i|}$



(d) $\kappa_y = {\max_i |y_i|}/{\min_i |x_i|}$

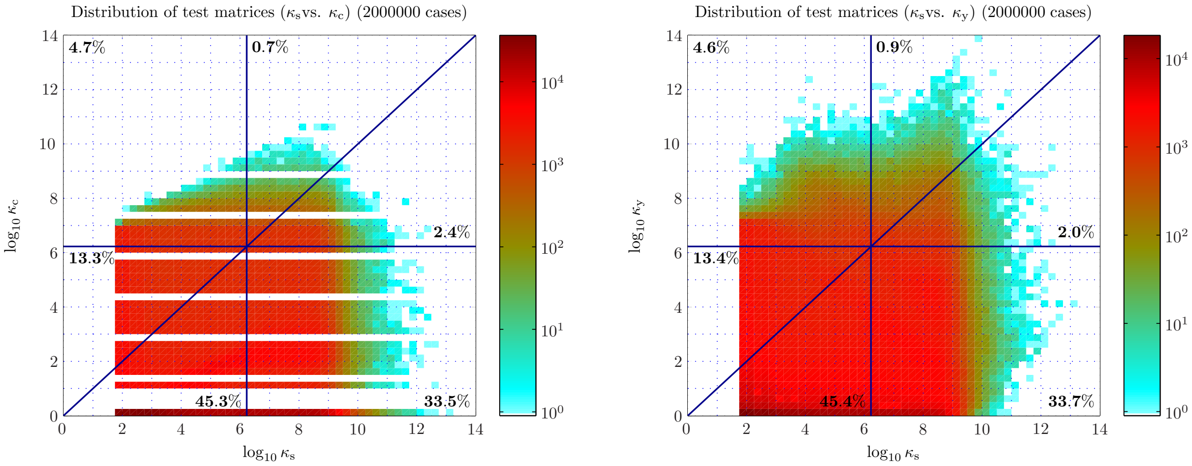


(e) $\kappa_{norm} = \kappa(RA)$



(f) $\kappa_{comp} = \kappa(RA\,\mathrm{diag}(x))$

Figure 2: Histograms of various properties of test matrices. The vertical blue line is located at ${1}/{\gamma\varepsilon_w}$, the threshold between "well-conditioned" and "ill-conditioned". The percentage of the test sample that belong to each category is indicated at the top near the vertical blue line. Here $\gamma = \max\{10, \sqrt{n}\} = 10$.



(a) κ_{s} vs. κ_{c} (b) κ_{s} vs. κ_{y}

Figure 3: Distribution of test matrices (2D histograms).

$1/\gamma\varepsilon_w$. These are matrices that are extremely ill-conditioned, and therefore we cannot guarantee accurate solutions. Of the two million test matrices, 1178903 cases fall into this category.

Note that the choice of $1/\gamma\varepsilon_w$ (which is approximately 1.67×10^6 for 100×100 matrices) as the separation between well and ill-conditioned matrices is somewhat arbitrary; we could have chosen a more conservative criteria such as $1/n\varepsilon_w$ or more aggressive criteria such as $1/\varepsilon_w$. Our data in Section 6 indicate that the choice $1/\gamma\varepsilon_w$ seems to give reliable solutions without throwing away too many convergent cases. The lower bound provided by $\max\{10, \sqrt{n}\}$ both protects against condition number underestimates and also keeps the bounds attainable for small systems.

Figure 2(f) shows the histogram of κ_{comp}, varying from about 57 to 4.5×10^{15}. Recall that this is the condition number for componentwise problem: $\kappa_{\mathrm{comp}} = \kappa(RA\,\mathrm{diag}(x)) = \kappa(A_s\,\mathrm{diag}(y)) \leq \kappa_{\mathrm{s}}\kappa_{\mathrm{y}}$. Note that κ_{comp} depends not only on the matrix A, but also on the right hand side vector b (since it depends on the solution x). As in the case of condition number for the normwise problem, we divide the problems into two categories according to κ_{comp}:

- **Componentwise well-conditioned problems**. These are problems with $\kappa_{\mathrm{comp}} \leq 1/\gamma\varepsilon_w$, and perhaps more accurately described as "not too ill-conditioned" problems. These are problems where we hope to have an accurate solution in componentwise sense. Of the two million test problems, 545427 cases fall into this category.

- **Componentwise ill-conditioned problems**. These are the rest of the problems with $\kappa_{\mathrm{comp}} > 1/\gamma\varepsilon_w$. These are problems that are extremely ill-conditioned, and therefore we cannot guarantee accurate solutions. Of the two million test problems, 1454573 cases fall into this category.

Note that if any component of the solution is zero, then κ_{comp} becomes infinite.

Figure 3(a) shows the distribution of the 2 million test matrices displayed as a 2D histogram, with the horizontal axis indicating the equilibrated condition number κ_{s} and the vertical axis indicating the condition number κ_{c} of the column scaling matrix C. Since these types of 2D

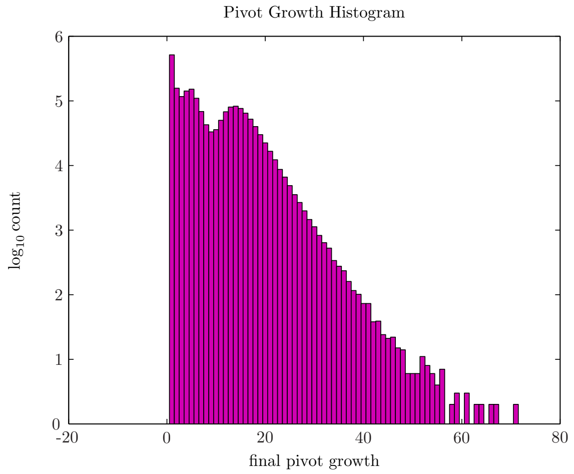


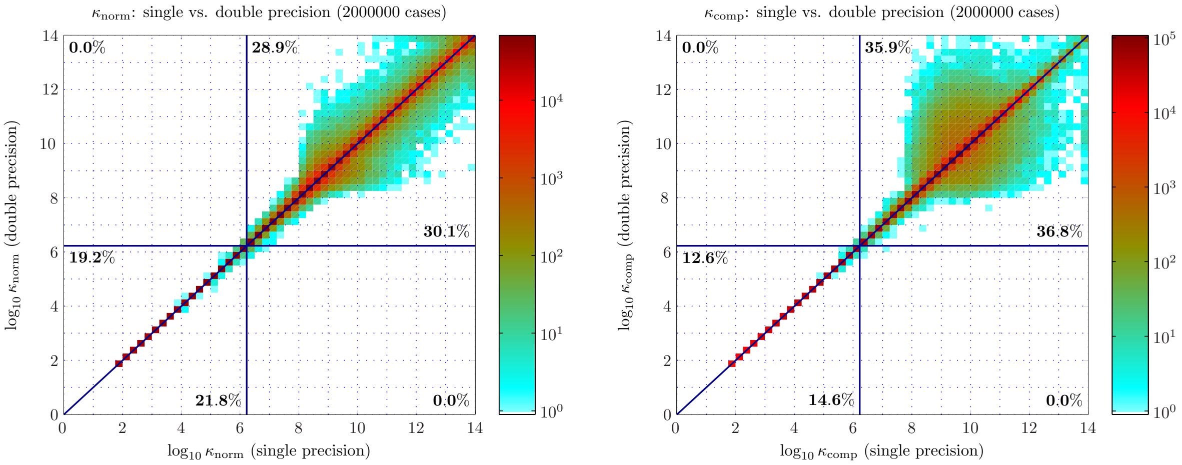
Figure 4: Histogram of GEPP pivot growth.

histograms appear throughout this paper, we explain this type of plot in some detail here. In each 2D histogram, each colored square at coordinate (x, y) indicates the existence of matrices that have κ_s in the range $[10^x, 10^{x+1/4})$ and κ_c in the range $[10^y, 10^{y+1/4})$. The color of each square indicates the number of matrices that fall in that square, indicated by the color bar to the right of the plot. Red (dark) colored squares indicate large population while cyan (light) colored squares indicate very small population. Note that logarithmic scale is used in the color bar, so a lighter color indicates a vastly smaller population than a darker color. For example, the light cyan colored squares near the top and left edges contains only a handful of matrices (less than 5, usually just 1), while the darker red colored squares contains approximately 10^3 to 10^4 samples. These graphs can be interpreted as a test matrix population density at each coordinate, with a resolution of $1/4 \times 1/4$ squares.

The blue horizontal and vertical lines are located at $\kappa_\mathrm{s} = 1/\gamma\varepsilon_w$ and $\kappa_\mathrm{c} = 1/\gamma\varepsilon_w$, respectively, separating "well-conditioned" and "ill-conditioned" matrices according to appropriate measure. Along with the diagonal line $\kappa_\mathrm{s} = \kappa_\mathrm{c}$, this separates the plot into six regions, and the percentage of samples in each region is displayed in bold numbers in the plot. So for example, 33.5% of test samples had $\kappa_\mathrm{s} > 1/\gamma\varepsilon_w$ and $\kappa_\mathrm{c} \le 1/\gamma\varepsilon_w$.

Similarly, figure 3(b) shows the distribution of the test matrices displayed as a 2D histogram, with the horizontal axis indicating the equilibrated condition number κ_s and the vertical axis indicating the condition number κ_y of the scaled solution y. Both figures 3(a) and 3(b) shows that we sampled the respective test matrix space thoroughly.

For the basic solution method, we used LAPACK's `sgetrf` (GEPP, $PA_s = LU$) and `sgetrs` (triangular solve) routines. We observed that the pivot growth factors $\max_{i,j} |U(i,j)| / \max_{i,j} |A_s(i,j)|$ are no more than 72. This implies that we have obtained LU factors with small normwise backward error. Figure 4 shows the histogram of the pivot growth. This is consistent with the results by Trefethen and Schreiber [32]. Their statistical analysis and empirical data suggested that for various distribution of random matrices, the average growth factor (normalized by the standard deviation of the initial elements) is about $n^{2/3} \approx 22$ for partial pivoting.



(a) κ_{norm} (b) κ_{comp}

Figure 5: Accuracy of computed condition numbers: single precision vs. double precision. The single precision κ_{comp} are those computed by Algorithm 3 with $\rho_{\text{thresh}} = 0.5$.

5.4 Accuracy of Single Precision Condition Numbers

All the condition numbers in section 5.3 were computed in double precision, which can be regarded as the truth. Since the actual code `sgesvxx` will only have access to single precision condition number, it is important to make sure that single precision result is accurate for condition numbers up to $1/\gamma\varepsilon_w$.

The 2D histograms of the normwise condition number κ_{norm} computed in single precision and double precision, is shown in figure 5(a). These 2D histograms were described in Section 5.3 (in the paragraph describing figure 3(a) on page 23). The horizontal axis displays the condition number computed in single, which is what our code `sgesvxx` computes to determine the trustworthiness of our solution. The vertical axis displays the same condition number computed in double precision (using extra-precision iterative refinement), which we regard as truth. We see that for those matrices with $\kappa_{\text{norm}} \leq 1/\gamma\varepsilon_w$, the single precision result matches the double precision result very closely, resulting in the dark red diagonal band in the bottom-left quadrant of the figure. This tells us that we can trust the single precision κ_{norm} to separate the not too ill-conditioned matrices from very ill-conditioned matrices.

Figure 5(b) tells the same story for the componentwise condition number κ_{comp}; the single precision κ_{comp} is accurate unless the problem is extremely ill-conditioned. Note that κ_{comp} depends on the computed solution (since $\kappa_{\text{comp}} = \kappa(A_s \operatorname{diag}(y))$); the displayed plot is for our algorithm (Algorithm 3) with $\rho_{\text{thresh}} = 0.5$. However, results with other values for the parameter ρ_{thresh} did not significantly alter the picture.

5.5 Testing Platforms

We have tested the code on two platforms: Sun UltraSPARCs running Solaris and Itanium 2 running Linux. The numerical results presented in this report are obtained using the 1.3 GHz Itanium II processors in the Berkeley CITRIS cluster. The iterative refinement code is written in Fortran and is driven by test code in C. The result-generating codes are compiled with the GNU compilers `g77` and `gcc`*. ATLAS 3.6.0 [34] provides the BLAS routines and the LAPACK factorization routine. The XBLAS reference implementation [17] plus the additional routines above provide our extended-precision capabilities. On the Itanium 2 platforms, we have also tested with Intel's Math Kernel Library 7.2 [9] and their version 8.0 C and Fortran compilers. On the UltraSPARC platform, tests were run with Sun's compilers and Performance Library 6.0 [21]. The statistics on each platform and BLAS library were approximately the same. The only differences occurred with ill-conditioned problems.

6 Numerical Results

In this section, we present the numerical results for our new algorithm, Algorithm 3. The statistics are based on two million 100×100 test matrices described in Section 5. Similar results were obtained on two million each of 5×5 and 10×10 matrices and also two hundred thousand 1000×1000 matrices†. In particular, we will use these examples to evaluate and justify the new algorithmic features by comparing to Wilkinson's scheme (Algorithm 4) and to the current LAPACK algorithm with our small modifications (Algorithm 5).

The normwise and componentwise true errors are denoted E_{norm} and E_{comp}, respectively. They are defined by

$$E_{\text{norm}} = \frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \quad \text{and} \quad E_{\text{comp}} = \max_i \frac{|x_i - \hat{x}_i|}{|x_i|},$$

where x is the true solution and $\hat{x}$ is the solution computed by the algorithm. Similarly, the normwise and componentwise error bounds (computed by the algorithms) are denoted B_{norm} and B_{comp}, respectively.

The presentation of normwise and componentwise results will be treated separately in Sections 6.1 and 6.2. In these two sections, we set the parameter $\rho_{\text{thresh}} = 0.5$ in Algorithms 3 and 4, because this value was historically used in Algorithm 4. The parameter i_{thresh} (maximum number of iterations allowed) is set very large so that the algorithm will not stop prematurely.

We now define some notation common to Sections 6.1 and 6.2. Both 2D histograms and tables will distinguish three cases, depending on the size of the true error or the error bound. For example, for E_{norm} we distinguish

1. **Strong Convergence**: $E_{\text{norm}} \le 2\gamma\varepsilon_w = 2\max\{10, \sqrt{n}\} \cdot \varepsilon_w$. This is the most desirable case, where the true error is of order ε_w. For example, the lower solid horizontal blue line in Figure 6(a) (and in other analogous figures) is at $E_{\text{norm}} = 2\gamma\varepsilon_w$.

2. **Weak Convergence**: $2\gamma\varepsilon_w < E_{\text{norm}} \le \sqrt{\varepsilon_w}$. We could not get strong convergence, but we did get at least half of the digits correctly. For example, the upper solid horizontal blue line in Figure 6(a) (and in other analogous figures) is at $E_{\text{norm}} = \sqrt{\varepsilon_w}$.

3. **No Convergence**: $E_{\text{norm}} > \sqrt{\varepsilon_w}$. We could not get a meaningful result.

*Both `g77` and `gcc` are from GNU compiler collection version 3.3.4, Debian 1:3.3.4-13

†Full set of plots summarizing the result of Algorithm 3 (with various ρ_{thresh} values), Algorithm 4, and Algorithm 5 on all 6.2 million matrices are available at `http://www.cs.berkeley.edu/~demmel/itref-data/`.

In addition, we often indicate the value of ε_w in figures as well. For example the dashed horizontal blue line in Figure 6(a) (and in other analogous figures) is at $E_{\text{norm}} = \varepsilon_w$.

Many other analogous figures have two solid horizontal and one dashed horizontal line to indicate similar thresholds for B_{norm}, E_{comp}, and B_{comp}.

In the final version of the code we set the error bound to 1 whenever its computed value exceeds $\sqrt{\varepsilon_w}$, in order to indicate that it did not converge according to our criterion above. But in this section we report the computed error bounds without setting them to 1, in order to better understand their behavior.

Later in Section 6.4 we will vary the parameters ρ_{thresh} and i_{thresh} to study how the behavior of Algorithm 3 changes. In particular, we will recommend "cautious" and "aggressive" values of i_{thresh} and ρ_{thresh}. The cautious settings, which we recommend as the default, yield maximally reliable error bounds for well-conditioned (or not too ill-conditioned) problems, and cause the code to report convergence failure on the hardest problems. The aggressive settings will lead to more iterations on the hardest problems and usually, but not always, give error bounds within a factor of 100 of the true error.

6.1 Normwise Error Estimate

In this section we look at the results in terms of normwise error and bound. We compare three algorithms: our new algorithm (Algorithm 3) with $\rho_{\text{thresh}} = 0.5$, Wilkinson's algorithm with our error bound formula (Algorithm 4), and the current LAPACK algorithm with our small modifications (Algorithm 5).

The most important observation is that both Algorithm 3 or Algorithm 4 deliver a tiny error (E_{norm} strongly converged) and a slightly larger error bound (B_{norm} also strongly converged) as long as $\kappa_{\text{norm}} < 1/\gamma\varepsilon_w$, i.e. for all not-too-ill-conditioned matrices in our test set (821097 out of 2 million). This is the best possible behaviour we could expect, and helps justify our recommendation for "cautious" use of Algorithm 3 in Section 6.4.

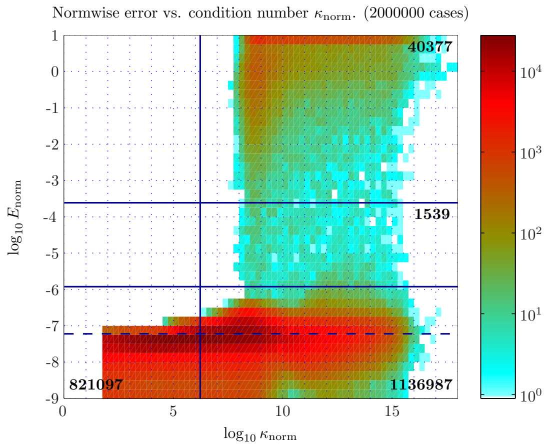
The second important observation is that for the harder problems, those with $\kappa_{\text{norm}} \geq 1/\gamma\varepsilon_w$, both algorithms also do very well, with Algorithm 3 getting strong convergence in both E_{norm} and B_{norm} in 96.4% of the cases, and Algorithm 4 getting strong convergence in both E_{norm} and B_{norm} in 92.3% of the cases.

In the rest of this section, we explore the experimental data in more detail, describing what goes wrong when we fail to get strong convergence.

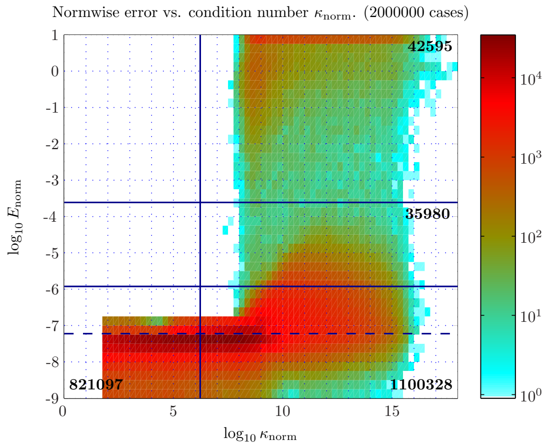
The three plots in Figure 6 show the 2D histograms of the test problems plotted according to their true normwise error E_{norm} and condition number κ_{norm} for the three algorithms. These 2D histograms were described in Section 5.3 on page 23; these plots can be interpreted as the population density of the test matrices. The vertical solid line is at $\kappa_{\text{norm}} = 1/\gamma\varepsilon_w$, and separates the not too ill-conditioned problems from the extremely ill-conditioned problems. Note that if any problem falls outside of the coordinate range, then it is placed at the edge; for example the band at the very top of Figure 6(a) are all the cases where $E_{\text{norm}} \geq 10$.

The first important conclusion to draw from Figure 6 is that for not-too-ill-conditioned problems ($\kappa_{\text{norm}} < 1/\gamma\varepsilon_w$), both Algorithm 3 and Algorithm 4 (Figures 6(a) and 6(b)) attain the best possible result: strong convergence of E_{norm} in all cases (821097 out of 2 million).

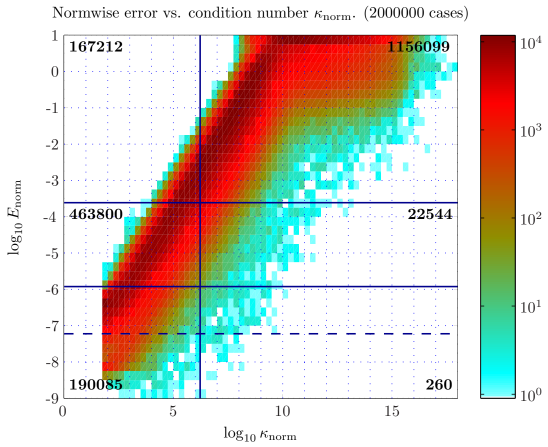
The second important conclusion is that for harder problems, with $\kappa_{\text{norm}} \geq 1/\gamma\varepsilon_w$ (1178903 cases) both Algorithm 3 and Algorithm 4 still do very well, with Algorithm 3 exhibiting strong



(a) Algorithm 3 with $\rho_{\text{thresh}} = 0.5$



(b) Algorithm 4 (Wilkinson)



(c) Algorithm 5 (LAPACK)

Figure 6: Normwise error vs. κ_{norm}.

convergence of E_{norm} in 96.4% of cases (1136987 out of 1178903), and Algorithm 4 exhibiting strong convergence of E_{norm} in 93.3% of cases (1100328 out of 1178903).

In contrast, with Algorithm 5 (Figure 6(c)), the error grows roughly proportional to the condition number, as shown by the dark diagonal squares in the figure. This is consistent with the early error analysis on the working precision iterative refinement [13, Theorem 12.2]. Algorithm 5 consistently gives much larger true errors and error bounds than either of the other two algorithms, when it converges.

But of course a small error E_{norm} is not helpful if the algorithm cannot recognize it by computing a small error bound B_{norm}. We now compare B_{norm} to E_{norm} to see how well our error estimate approximates the true error. Figure 7 shows the value of $(E_{\text{norm}}, B_{\text{norm}})$ for each problem, as a 2D histogram. The plots on the left includes all two million test cases, while the plots on the right only include the not-too-ill-conditioned problems, those with $\kappa_{\text{norm}} \leq {}^{1}/{\gamma\varepsilon_w}$. The vertical solid blue lines are at $E_{\text{norm}} = 2\gamma\varepsilon_w$ (corresponding to the threshold for strong convergence) and $E_{\text{norm}} = \sqrt{\varepsilon_w}$ (corresponding to the threshold for weak convergence). Horizontal blue lines are at B_{norm} equal to the same values. The diagonal line marks where B_{norm} is equal to E_{norm}; matrices below the diagonal correspond to *underestimates* ($B_{\text{norm}} < E_{\text{norm}}$), and matrices above the diagonal correspond to *overestimates* ($B_{\text{norm}} > E_{\text{norm}}$), The vertical and horizontal dotted lines correspond to $E_{\text{norm}} = \varepsilon_w$ and $B_{\text{norm}} = \varepsilon_w$, respectively.

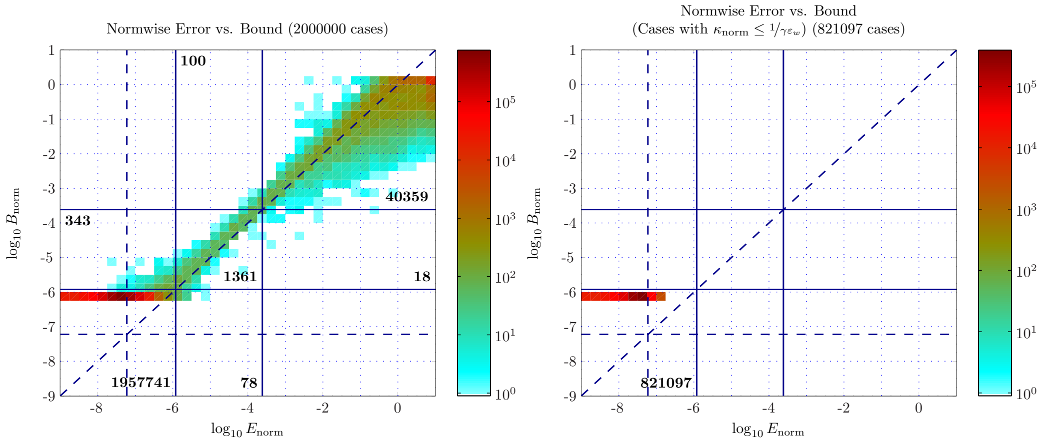
First consider the right-hand plots in Figure 7(a) and Figure 7(b). These show the happy result that for not-too-ill-condition problems, both Algorithm 3 and Algorithm 4 perform perfectly: strong convergence in both E_{norm} and B_{norm} in all cases. Furthermore, B_{norm} always slightly overestimates E_{norm}. Thus we can trust either algorithm to deliver a tiny error and a slightly larger error bound as long as $\kappa_{\text{norm}} < {}^{1}/{\gamma\varepsilon_w}$.

Now consider the left-hand plots in the same figures. By subtracting out the 821097 not-too-ill-conditioned cases, we get the distribution of results $(E_{\text{norm}}, B_{\text{norm}})$ for all ill-conditioned cases. Most still yield strong convergence in both quantities, but it is interesting to contrast the behavior of Algorithms 3 and 4.

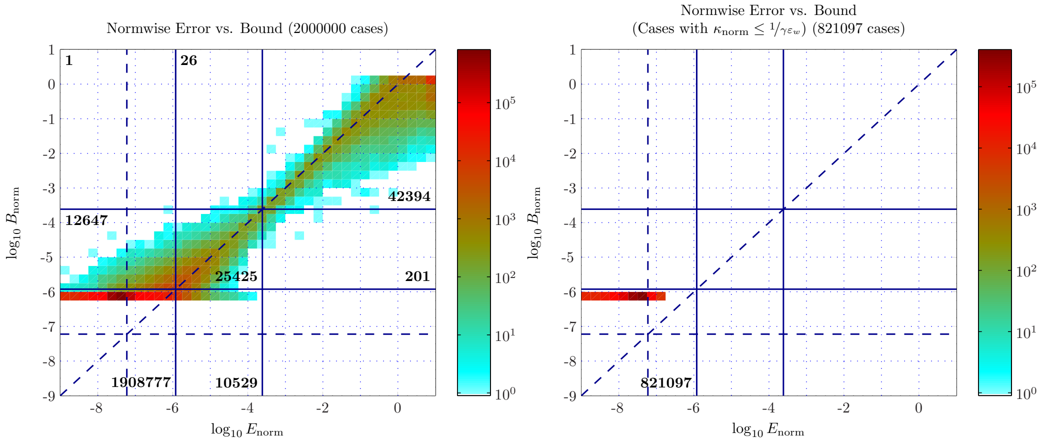
The first impression from these plots is that among these ill-conditioned cases, both Algorithms 3 and 4 fail to have B_{norm} converge in about same number of cases: 3.4% and 3.6% repectively. But of the remainder, those where both algorithms report an error bound to the user, Algorithm 3 fails to get strong convergence in both E_{norm} and B_{norm} in only 0.16% of the cases (1800 out of 1138444 cases), whereas Algorithm 4 fails to get strong convergence in both E_{norm} and B_{norm} over 27 times as often, in 4.3% of the cases (48802 out of 1136482). In other words, there are over 27 times more cases where Algorithm 4 is "confused" than Algorithm 3 (48802 versus 1800), and may return significantly disagreeing E_{norm} and B_{norm} (see also Figures 8(a) and 8(b)).

To further understand the difference between Algorithms 3 and 4, we further classify all 2 million cases according to whether the entries of the 4-tuple (E_{norm}(Alg 3), B_{norm}(Alg 3), E_{norm}(Alg 4), B_{norm}(Alg 4)) are strongly converged, weakly converged or not converged, i.e. $3^4 = 81$ categories in all. This results in the 9×9 table shown in Table 1. For example, the entry at bottom-left indicates that 1906992 cases achieved strong convergence in both E_{norm} and B_{norm} for both Algorithms 3 and 4, and the top-right entry indicates that in 38705 cases both algorithms failed to converge in either E_{norm} or B_{norm}.

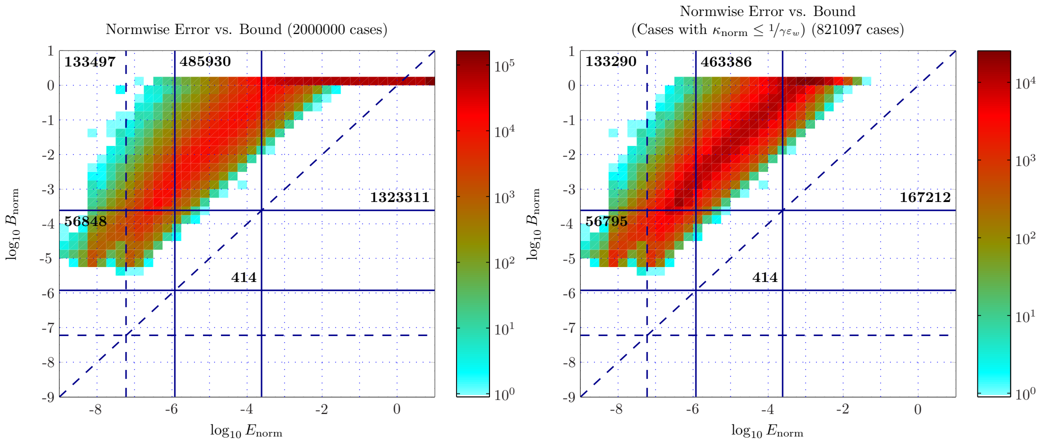
Eliminating these 1906992+38705 = 1945697 cases leaves 54303 cases (all ill-conditioned) where Algorithms 3 and 4 behaved significantly differently. Of these 54303 cases, Algorithm 3 significantly outperformed Algorithm 4, getting strong convergence in E_{norm} and B_{norm} 93% of the time, versus



(a) Algorithm 3 with $\rho_{\text{thresh}} = 0.5$



(b) Algorithm 4 (Wilkinson)



(c) Algorithm 5 (LAPACK)

Figure 7: Normwise error vs. bound. The left plots include all two million cases, while the right plots include only the well-conditioned ($\kappa_{\text{norm}} < 1/\gamma\varepsilon_w$) cases.

		E_{norm} in Alg. 3								
		strong			weak			fail		
B_{norm} in Alg. 3	fail					3	41		2	38705
					1	31	4	21	320	50
					19	1		1215	46	
	weak		6	25	1	9	258			1
		7	227	8	15	534	28			16
		63	7		485	31		1		
	strong		6	3363			1			
		12597	24255	94	6	58	1			
		1906992	10434		2	10				

Table 1: Classification of matrices using normwise measure. The matrices are first classified according to Algorithm 3 results, resulting in the 3×3 blocks delineated by solid lines. Then in each of the 9 boxes, the result is further classified according to the result of Algorithm 4, with columns and rows ordered in the same fashion. Note that each blocks are arranged so that it matches the location found in Figure 7.

3% for Algorithm 4.

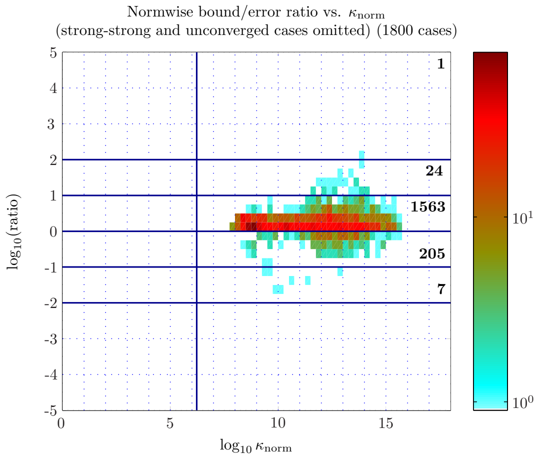
Finally, Figure 8 shows the 2D histogram of the ratio $B_{\text{norm}}/E_{\text{norm}}$ plotted against κ_{norm}. These plots show how much B_{norm} overestimates E_{norm} (ratio > 1) or underestimates E_{norm} (ratio < 1). We omit cases where B_{norm} does not converge, and also cases where both E_{norm} and B_{norm} converged strongly (the ideal case), since we are only interested in analyzing in cases where the algorithm claims to converge to a solution but either E_{norm} or B_{norm} or both are much larger than ε_w. Since Algorithms 3 and 4 converge strongly for both E_{norm} and B_{norm} for all not-too-ill-conditioned cases, no data points appear left of the vertical line in Figures 8(a) and 8(b). We are most concerned about underestimates, where the error bound is substantially smaller than the true error. We see that Algorithm 3 has somewhat fewer underestimates than Algorithm 4 (defined as $E_{\text{norm}} > 10B_{\text{norm}}$): 7 vs. 243 and rather fewer overestimates ($10E_{\text{norm}} < B_{\text{norm}}$), 25 vs. 2130.

Figure 8(c) indicates that while Algorithm 5 never underestimates the error, it almost always overestimates the error by two or three orders of magnitude. Combined with Figure 7(c) indicating that no example converged strongly, the error bound returned by Algorithm 5 is not too practical, albeit being safe.

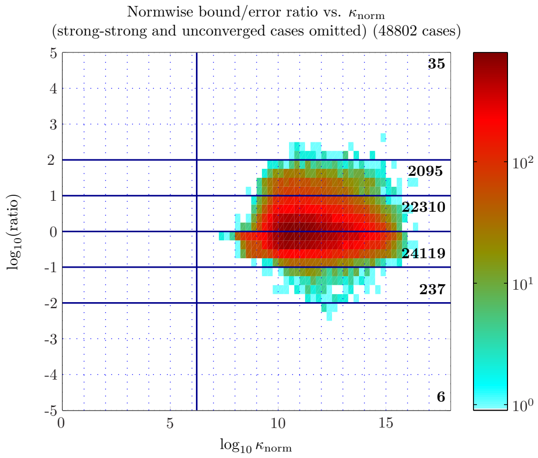
6.2 Componentwise Error Estimate

In this section we look at the results in terms of componentwise true error E_{comp} and error bound B_{comp}. We compare three algorithms: our new algorithm (Algorithm 3) with $\rho_{\text{thresh}} = 0.5$, Wilkinson's algorithm (Algorithm 4), and the current LAPACK algorithm with our modifications to compute a componentwise error bound (Algorithm 5). Algorithm 4 does not compute a componentwise error bound, nor was it designed to make E_{comp} small, so only its true error E_{comp} is compared in this section.

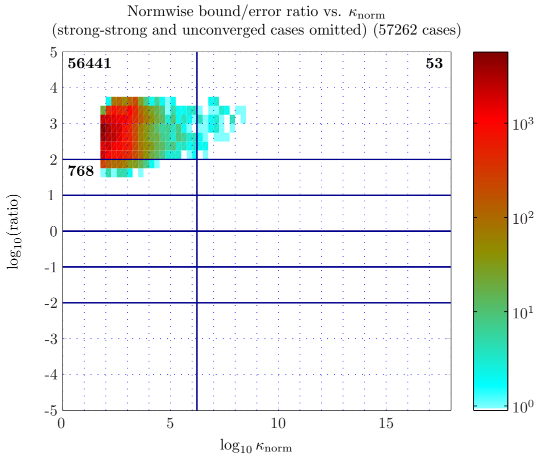
The most important observation is that Algorithm 3 delivers a tiny componentwise error (E_{comp} strongly converged) and a slightly larger error bound (B_{comp} also strongly converged) as long as $\kappa_{\text{comp}} < 1/\gamma\varepsilon_w$, i.e. for all not-too-ill-conditioned matrices in our test set (that is, not too ill-conditioned with respect to the componentwise condition number κ_{comp}), 545427 out of 2 million



(a) Algorithm 3 with $\rho_{\text{thresh}} = 0.5$



(b) Algorithm 4 (Wilkinson)



(c) Algorithm 5 (LAPACK)

Figure 8: Overestimation and underestimation ratio ($B_{\text{norm}}/E_{\text{norm}}$) vs. κ_{norm}. Cases with strong convergence (in both E_{norm} and B_{norm}) and cases with no convergence ($B_{\text{norm}} > \sqrt{\varepsilon_w}$) are omitted for clarity.

cases. This is the best possible behavior we could expect, and helps justify our recommendation for "cautious" use of Algorithm 3 in Section 6.4.

The second important observation is that for the harder problems, those with $\kappa_{\text{comp}} \geq {}^1\!/\gamma\varepsilon_w$, Algorithm 3 also does very well, getting strong convergence in both E_{comp} and B_{comp} in 94% of the cases.

In the rest of this section, we explore the experimental data in more detail, describing what goes wrong when we fail to get strong convergence.

The three plots in Figure 9 show the 2D histograms of the test problems plotted according to their componentwise error E_{comp} and condition number κ_{comp} for the three algorithms. These graphs may be interpreted similarly to those in Figure 6, which were described in the last section.

The first important conclusion to draw from Figure 9 is that for not-too-ill-conditioned problems ($\kappa_{\text{comp}} < {}^1\!/\gamma\varepsilon_w$), Algorithm 3 attains the best possible result: strong convergence of E_{comp} in all cases (all 545427 out of 2 million). Algorithm 4, which was not designed to get small componentwise errors, does slightly worse, with strong convergence in 99% of the cases (539342 out of 545427), and weak or no convergence in the other 1%, including a few really well-conditioned problems.

The second important conclusion is that for harder problems, with $\kappa_{\text{comp}} \geq {}^1\!/\gamma\varepsilon_w$ (1454573 cases) Algorithm 3 still does very well, exhibiting strong convergence of E_{comp} in 95% of cases (1380569 out of 1454573). Algorithm 4 does worse, exhibiting strong convergence of E_{comp} in only 66.6% of cases (968198 out of 1454573), and failing to converge at all more than twice as frequently (104108 versus 41755).

In contrast, with Algorithm 5 the error grows roughly proportional to the condition number, as shown by the dark diagonal squares in the figure. Strong convergence of E_{comp} is very rare, only 7.3% of not-too-ill-conditioned cases, and not at all for ill-conditioned cases.

As in the last section, we note that a small error E_{comp} is helpful only if the algorithm also produces a comparably small error bound B_{comp}. Consider Figure 10, whose interpretation is the same as that of Figure 7 in the last section.

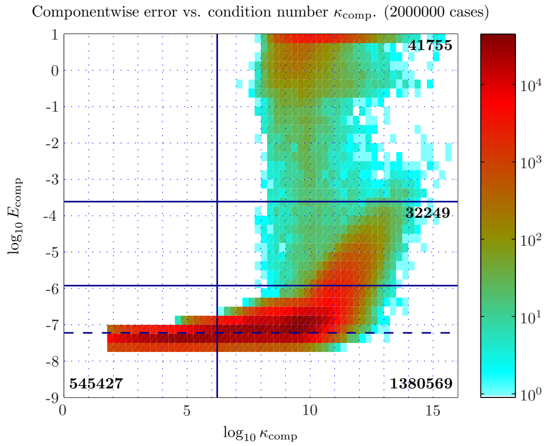
First consider the right-hand plot in Figure 10(a). This shows the happy result that for not-too-ill-conditioned problems, Algorithm 3 performs perfectly: strong convergence in both E_{comp} and B_{comp} in all cases. Furthermore, B_{comp} always slightly overestimates E_{comp}. Thus we can trust Algorithm 3 to deliver a tiny error and a slightly larger error bound as long as $\kappa_{\text{comp}} < {}^1\!/\gamma\varepsilon_w$.

Now consider the left-hand plot in the same figure. By subtracting out the 545427 not-too-ill-conditioned cases, we get the distribution of results $(E_{\text{comp}}, B_{\text{comp}})$ for all ill-conditioned cases. Most still yield strong convergence in both quantities (94%). In contrast, Algorithm 5 never converges strongly.

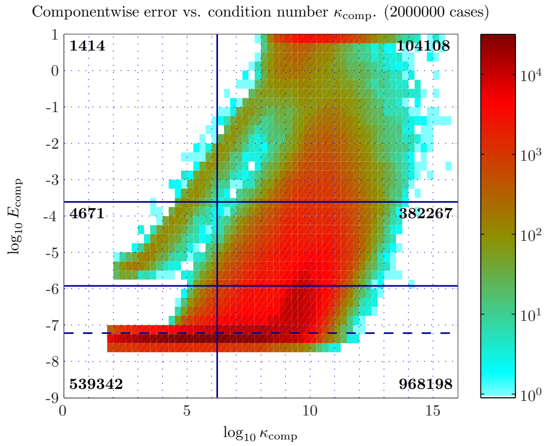
Finally, Figure 11 shows the 2D histogram of the ratio ${}^{B_{\text{comp}}}\!/_{E_{\text{comp}}}$ plotted against κ_{comp}. This graph is very similar to those in Figure 8, which were described in the last section. In contrast to that figure, we see there are more cases where Algorithm 3 attains neither strong convergence in both true error and error bound, nor convergence failure in both: 45100 cases versus 1800 (both out of 2 million, so rather few either way). Considering underestimates of the error, we see that there are also more cases where the ratio is less than 0.1, 273 vs. 7.

6.3 Iteration Counts and Running Time

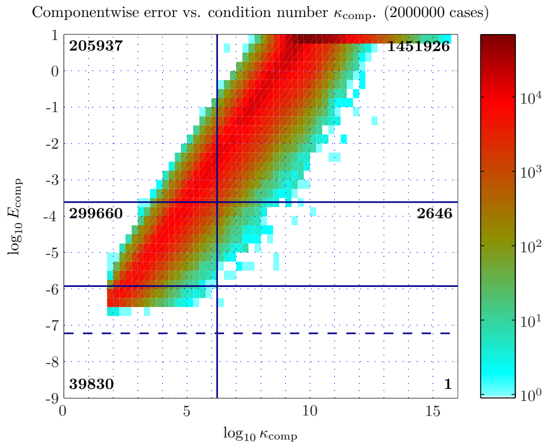
Figure 12 shows the relation between number of iterations and the componentwise condition number κ_{comp}. We see that for well conditioned problems, all three algorithms require less than 5 iterations.



(a) Algorithm 3 with $\rho_{\text{thresh}} = 0.5$

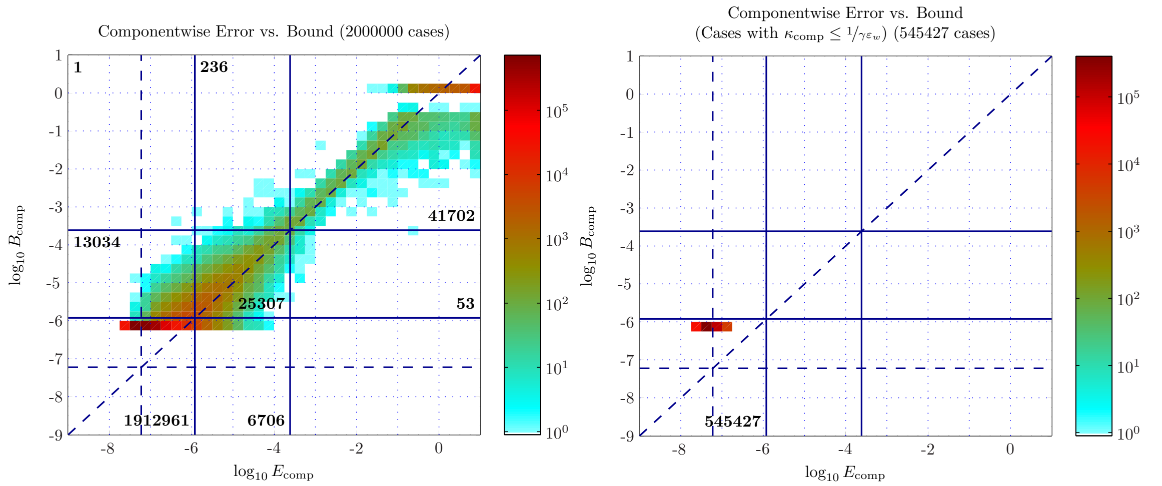


(b) Algorithm 4 (Wilkinson)

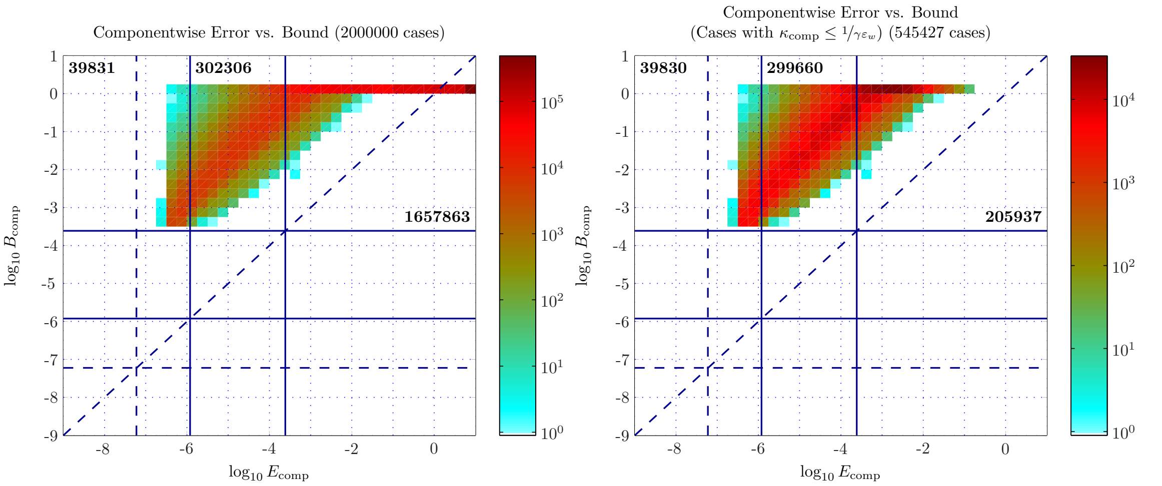


(c) Algorithm 5 (LAPACK)

Figure 9: Componentwise error vs. κ_{comp}.



(a) Algorithm 3 with $\rho_{\text{thresh}} = 0.5$



(b) Algorithm 5 (LAPACK)

Figure 10: Componentwise error vs. bound. The left plot includes all two million cases, while the right plot includes only the well-conditioned ($\kappa_{\text{comp}} < {}^1/_{\gamma\varepsilon_w}$) cases.

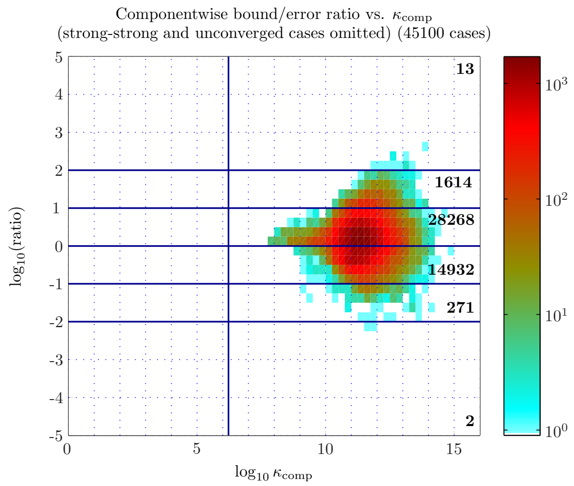


Figure 11: Overestimation and underestimation ratio ($^{B_{\mathrm{comp}}}/_{E_{\mathrm{comp}}}$) vs. κ_{comp}, Algorithm 3 with $\rho_{\mathrm{thresh}} = 0.5$. Cases with strong convergence (in both E_{comp} and B_{comp}) and cases with no convergence ($B_{\mathrm{comp}} > \sqrt{\varepsilon_w}$) are omitted for clarity.

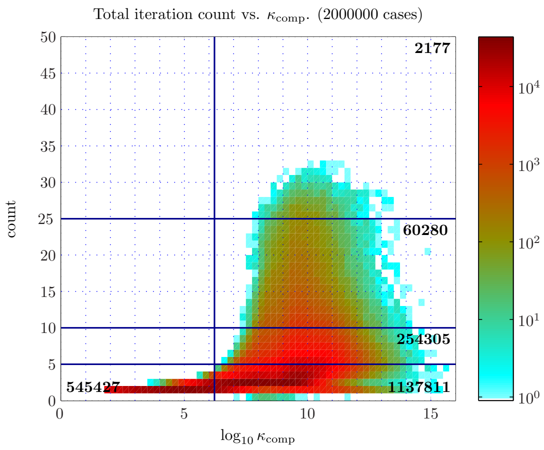
The statistical summary of the iteration count (broken down into single x and doubled-x iterations) is shown in Table 2. Note that Figure 12(a) is for $\rho_{\mathrm{thresh}} = 0.5$. Other values of ρ_{thresh} in Table 2 is discussed further in Section 6.4.2.

We now look at the number of cases where doubled-x scheme was used (in Algorithm 3 with $\rho_{\mathrm{thresh}} = 0.5$). Of the 545427 cases with well-conditioned componentwise condition number, in 299984 (55%) cases the doubled-x scheme was triggered. A maximum of 3 iterations in doubled-x was performed (in 649 cases), with an average of 0.66 iterations, compared to an average of 1.5 single-x iterations. Thus we see that for well-conditioned systems we do not spend too much time in the more expensive doubled-x scheme. Note that the average and maximum number of iterations in Algorithm 3 is identical to that of Wilkinson's Algorithm 4, and is less than that of LAPACK's Algorithm 5. Of the 1454573 cases with ill-conditioned componentwise condition number, almost all (99.994%) required the doubled-x scheme, and all but the very first iteration was done in doubled-x.

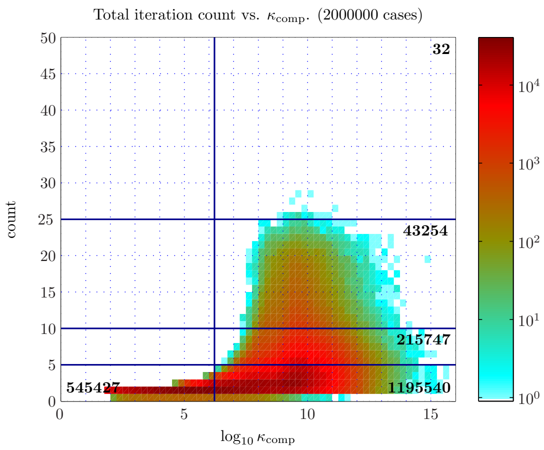
The doubled-x iteration performs a `gemv2` operation described in Section 5.1, which requires about twice the number of floating-point operations as in the normal `gemv` used in single-x iterations. However, since they require approximately same number of memory references, we expect with some optimizations the actual running time of `gemv2` will be much less than twice that of `gemv`.

6.4 Effects of various parameters in Algorithm 3

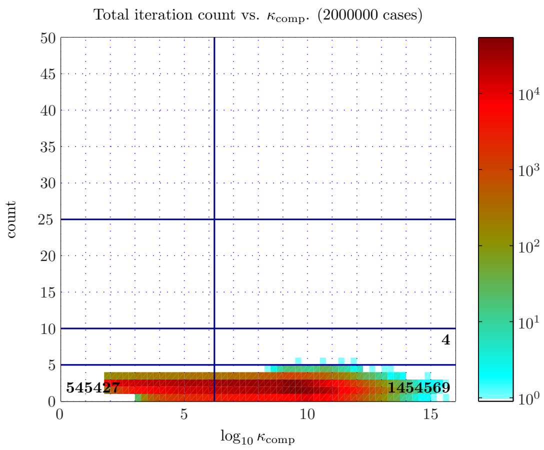
Compared to Algorithm 4, Algorithm 3 incorporates several new algorithmic ingredients and adjustable parameters. We note that different parameter settings in Algorithm 3 usually do not make any difference for the well-conditioned problems, since all of them quickly converge strongly. However they can make noticeable differences for the very ill-conditioned problems. In this section, we examine the effect of each individual parameter setting, using these difficult problems.



(a) Algorithm 3 with $\rho_{thresh} = 0.5$



(b) Algorithm 4 (Wilkinson)



(c) Algorithm 5 (LAPACK)

Figure 12: Total iteration count vs. κ_{comp}.

	single x			doubled x			total			doubled-x
	max	mean	med	max	mean	med	max	mean	med	incidence
Alg. 3 ($\rho_{\text{thresh}} = 0.5$)	3	1.5	1	3	0.7	1	4	2.1	2	55%
Alg. 3 ($\rho_{\text{thresh}} = 0.8$)	3	1.5	1	3	0.7	1	4	2.1	2	55%
Alg. 3 ($\rho_{\text{thresh}} = 0.9$)	3	1.5	1	3	0.7	1	4	2.1	2	55%
Alg. 3 ($\rho_{\text{thresh}} = 0.95$)	3	1.5	1	3	0.7	1	4	2.1	2	55%
Alg. 4 (Wilkinson)							4	2.1	2	
Alg. 5 (LAPACK)							4	2.6	3	

(a) Well-conditioned ($\kappa_{\text{comp}} \le 1/\gamma\varepsilon_w$)

	single x			doubled x			total			doubled-x
	max	mean	med	max	mean	med	max	mean	med	incidence
Alg. 3 ($\rho_{\text{thresh}} = 0.5$)	1	1	1	32	3.6	3	33	4.6	4	100%
Alg. 3 ($\rho_{\text{thresh}} = 0.8$)	1	1	1	89	4.0	3	90	5.0	4	100%
Alg. 3 ($\rho_{\text{thresh}} = 0.9$)	1	1	1	175	4.1	3	176	5.1	4	100%
Alg. 3 ($\rho_{\text{thresh}} = 0.95$)	1	1	1	330	4.3	3	331	5.3	4	100%
Alg. 4 (Wilkinson)							29	4.0	3	
Alg. 5 (LAPACK)							6	2.4	2	

(b) Ill-conditioned ($\kappa_{\text{comp}} > 1/\gamma\varepsilon_w$)

Table 2: Statistics (max, mean, and median) on the number of iterations required by each algorithm. "single x" refers to the iterations where x is kept in working precision (single), while "doubled x" refers to the iterations where x is kept in doubled working precision (doubled-single). Algorithms 4 and 5 does not have doubled-x scheme and hence some columns are left blank.

6.4.1 Effect of doubled-x iteration

For ill-scaled systems, the doubled-x iteration is very useful in order to get accurate results for the small components in the solution. We did the following experiment to support this statement. We ran Algorithm 3 with and without the doubled-x iteration for the two million test cases.

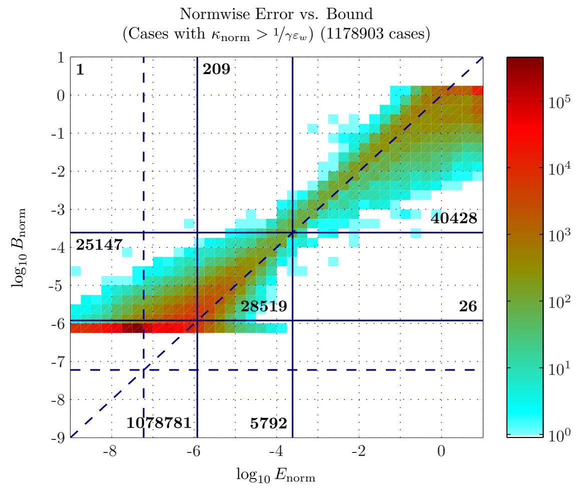
Figure 13 shows the convergence statistics for the two methods. With doubled-x iteration, Algorithm 3 obtains 57863 (2.9%) more cases of strong-strong normwise convergence as well as 256030 (12.8%) more cases of strong-strong componentwise convergence. The number of cases where the code reports normwise non-convergence ($B_{\text{norm}} > \sqrt{\varepsilon_w}$) decreases by 179; cases of componentwise non-convergence ($B_{\text{comp}} > \sqrt{\varepsilon_w}$) decreases by 5581.

6.4.2 Effect of ρ_{thresh}

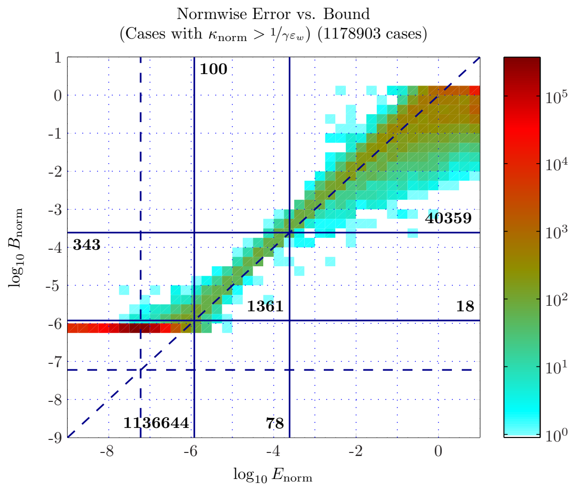
In Algorithm 3, ρ_{thresh} is one of the most important parameters that may affect the iteration behavior. It is used in Criterion (17) (see page 11) to determine when to stop the iteration:

$$\text{stop if} \quad \frac{\|dx^{(i+1)}\|_\infty}{\|dx^{(i)}\|_\infty} \ge \rho_{\text{thresh}}. \tag{23}$$

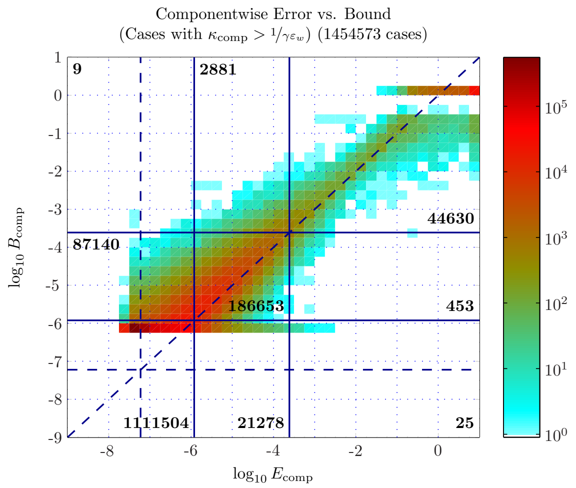
A larger ρ_{thresh} allows the algorithm to make progress more slowly and take more steps to converge. This is useful for some very ill-conditioned problems for which the iteration may terminate prematurely with a smaller ρ_{thresh}. However, a larger ρ_{thresh} may cause more severe overestimates



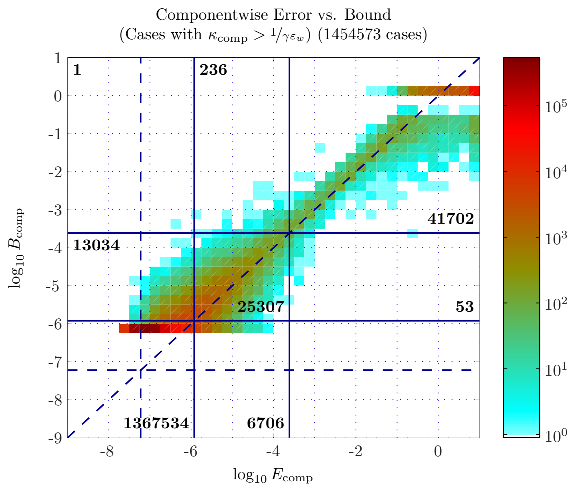
(a) Without doubled-x scheme (normwise)



(b) With doubled-x scheme (normwise)

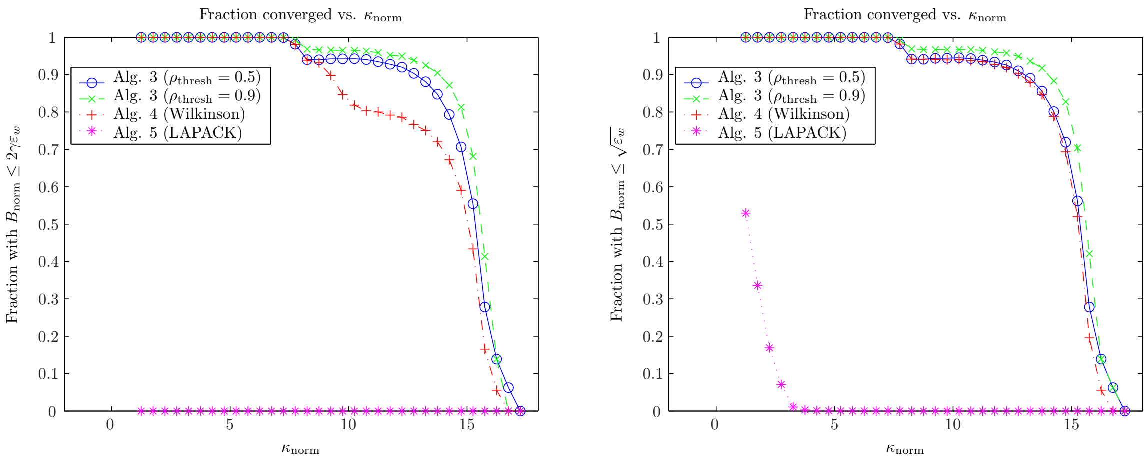


(c) Without doubled-x scheme (componentwise)



(d) With doubled-x scheme (componentwise)

Figure 13: Effect of doubled-x iteration. Only ill-conditioned problems ($\kappa_{\text{norm}} > 1/\gamma\varepsilon_w$ for normwise case and $\kappa_{\text{comp}} > 1/\gamma\varepsilon_w$ for componentwise case) are displayed.



(a) Strong convergence ($B_{\rm norm} \le 2\gamma\varepsilon_w$)

(b) Weak convergence ($B_{\rm norm} \le \sqrt{\varepsilon_w}$)

Figure 14: Fraction converged (based on normwise bound $B_{\rm norm}$) plotted against normwise condition number $\kappa_{\rm norm}$.

(because of the $(1-\rho_{\rm thresh})$ factor in the denominator of the error bound) and underestimates (since we are being more aggressive to pursue a small dx).

Figure 14 shows the convergence performance (measured by normwise bound) of various algorithms. We see that Algorithm 3 achieves strong convergence more often than Algorithm 4. Larger $\rho_{\rm thresh}$ in Algorithm 3 also makes some difference for ill-conditioned systems.

Table 3 gives the number of overestimates and underestimates of the error bounds returned by Algorithm 3, as a function of $\rho_{\rm thresh}$. We see that the number of unconverged cases drops nearly in half as we increase $\rho_{\rm thresh}$ from 0.5 to 0.95 at the cost of more severe overestimates and underestimates.

Table 2 displays the statistics of the total iteration counts for various algorithms. For well-conditioned problems, Algorithm 3 (with various $\rho_{\rm thresh}$) and Algorithm 4 all require about the same number of steps (maximum of 4 with median of 2). For ill-conditioned problems, Algorithm 4 requires slightly fewer iterations than Algorithm 3 (at the cost of not converging in some cases). Within Algorithm 3 it is clear that a larger $\rho_{\rm thresh}$ may potentially need a much larger number of iterations. However, a large number of iterations is required only when the problem is extremely hard and happens relatively rarely (hence the median stays at 4).

6.4.3 Justification of various components in the error bound

Figure 15 shows the true error $E_{\rm norm}$ versus the error bound $B_{\rm norm}$ when using the following formulas for computing $B_{\rm norm}$:

a. $B_{\rm norm} = \frac{\|dx\|_\infty}{\|x\|_\infty}$,

b. $B_{\rm norm} = \frac{\|dx\|_\infty}{\|x\|_\infty(1-\rho_{\rm max})}$,

c. $B_{\rm norm} = \max\left\{\frac{\|dx\|_\infty}{\|x\|_\infty(1-\rho_{\rm max})},\ \gamma\varepsilon_w\right\}$.

	Underestimates		Overestimates		No convergence
	$> 100\times$	$> 10\times$	$> 100\times$	$> 10\times$	
Alg. 3 with $\rho_{\text{thresh}} = 0.5$	0	7	1	25	40459
Alg. 3 with $\rho_{\text{thresh}} = 0.8$	0	30	3	151	25452
Alg. 3 with $\rho_{\text{thresh}} = 0.9$	0	34	3	505	22755
Alg. 3 with $\rho_{\text{thresh}} = 0.95$	0	33	14	843	21673
Alg. 4 (Wilkinson)	6	243	35	2130	42421
Alg. 5 (LAPACK)	0	0	56494	57262	1942738

(a) Normwise

	Underestimates		Overestimates		No convergence
	$> 100\times$	$> 10\times$	$> 100\times$	$> 10\times$	
Alg. 3 with $\rho_{\text{thresh}} = 0.5$	2	273	13	1627	41939
Alg. 3 with $\rho_{\text{thresh}} = 0.8$	5	463	36	3842	26847
Alg. 3 with $\rho_{\text{thresh}} = 0.9$	6	502	67	7436	24250
Alg. 3 with $\rho_{\text{thresh}} = 0.95$	8	499	140	11094	23297

(b) Componentwise

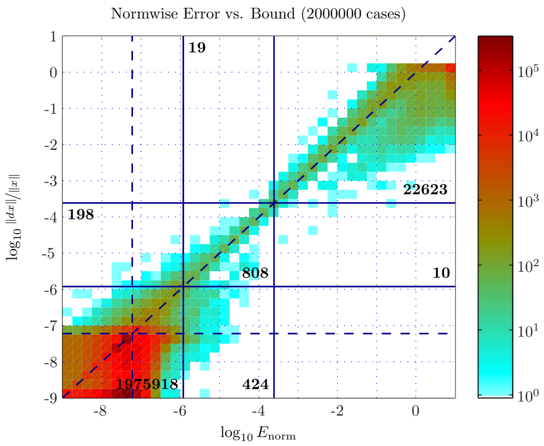
Table 3: Number of overestimates and underestimates of the error returned by various algorithms. Cases with strong convergence in both true error and error bound are not included in the underestimates and overestimates. The number of cases with no convergence is also listed. The category "$> 10\times$" includes the cases under "$> 100\times$".

The purpose of carefully choosing the definition of B_{norm} is to make it as reliable an upper bound on E_{norm} as possible. The 2D histograms in Figure 15 justify our choice of formula for B_{norm}. As we add more components to the error bound (from (a) to (b) to (c)), the number of severe underestimates ($> 100\times$) decreases from 38 to 6 to 0. However, the number of severe overestimates ($> 100\times$) increases from 0 to 3 to 3. We feel that reducing the number of severe underestimates is desirable even if it increases the number of overestimates by a modest amount.

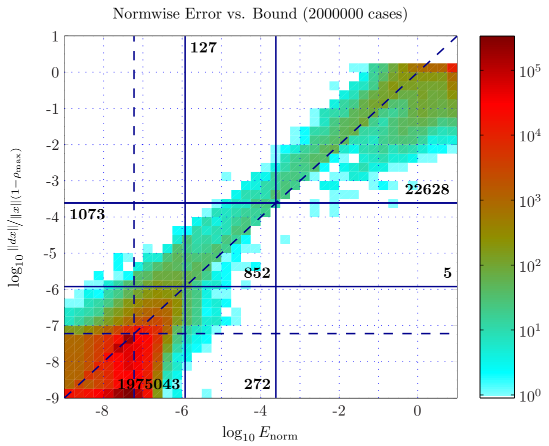
6.5 "Cautious" versus "aggressive" parameter settings

To summarize, by setting ρ_{thresh} and i_{thresh} smaller or larger, Algorithm 3 can be made "cautious" or "aggressive". The cautious parameter setting should be used for well-conditioned or not too ill-conditioned problems and we recommend this as the default setting in the algorithm. In this case, the algorithm always terminates quickly, and according to our statistical testing with 2×10^6 matrices, provides a reliable error bound. By examining the output reciprocal condition estimate (`rcond_nrm` or `rcond_cmp`) to see if it exceeds $1/\gamma\varepsilon_w$, the user can have high confidence in the computed error bounds. The cautious setting also works for a large fraction of the most ill-conditioned problems, achieving strong normwise convergence in 96.4% of cases and strong componentwise convergence in 94.0% of cases. Failure to converge is indicated by returning $B_{\text{norm}} = 1$ and/or $B_{\text{comp}} = 1$, meaning no accuracy is guaranteed. We expect that most users would prefer this cautious mode as the default.

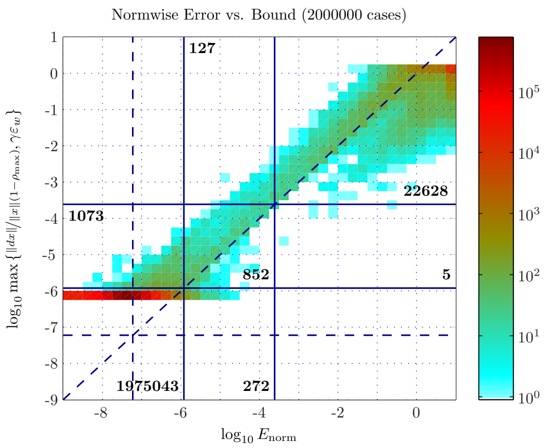
On the other hand, the aggressive parameter setting could be used for very ill-conditioned problems. In this mode, the algorithm is allowed to iterate much longer, and so more often arrives



(a) $\|dx\|_\infty/\|x\|_\infty$



(b) Multiply 15(a) by $1/(1-\rho_{\max})$



(c) Max 15(b) with $\gamma\varepsilon_w$

Figure 15: Effects of incorporating various components in B_{norm} for Algorithm 3 with $\rho_{\text{thresh}} = 0.9$.

	ρ_{thresh}	i_{thresh}
cautious	0.5	10
aggressive	0.9	100

Table 4: Recommended parameter settings for Algorithm 3.

at a fairly accurate solution. But there may be a number of cases that the returned error bound is not very reliable (either too large or too small.) The aggressive mode can work for a larger fraction of the extremely ill-conditioned problems.

Table 4 lists our recommended settings in the above two situations, based on our experimental data in Section 6.4. The cautious setting $\rho_{\text{thresh}} = 0.5$ was also used in the earlier literature.

7 Limitations of Refinement and our Bounds

The analysis in Section 2 and the algorithm in Section 3 rely on a few crucial assumptions. We assume that the system $Ax = b$ is not so ill-conditioned that iterative refinement fails to converge altogether. And we assume that the rounding errors in the residual (δr) and update (δx) computations are negligible until termination. For Section 2.3's componentwise estimates, we assume no entry of any computed solution or the true solution is exactly zero. These assumptions lead to limitations on Algorithm 3.

A strongly ill-conditioned system may produce a computed $\hat{x}$ that is far from the true x. Algorithm 3's error estimates may be quite incorrect when the error is large, as well. Correct but somewhat cautious guidelines for interpreting Algorithm 3's bounds are that

- the normwise error bound B_{norm} is unreliable when $\kappa_{\text{norm}} = \kappa_\infty(RA) \geq 1/\gamma\varepsilon_w$,
- the componentwise error bound B_{comp} is unreliable when $\kappa_{\text{comp}} = \kappa_\infty(RA\,\text{diag}(x)) \geq 1/\gamma\varepsilon_w$, and
- any error bound B_{norm} or B_{comp} is unreliable when it is at least $\sqrt{\varepsilon_w}$.

Condition numbers relative to application-specific, structured perturbations [7, 8, 25, 26, 33] should capture many of the successful cases our cautious settings forgo.

The rounding errors δr and δx affect the algorithm adversely only for ill-conditioned systems. The former is known, but the effect of δx on the componentwise convergence previously has not been discussed. Extra precision reduces both these effects, and monitoring particular condition numbers allows us to employ that extra precision selectively. Additionally, zero components in the solution may prevent componentwise convergence. Section 7.3 describes how Algorithm 3 correctly handles zero and tiny components.

7.1 Conditioning

Classical iterative refinement results guarantee convergence only when $\kappa_\infty(A)$ is sufficiently less than $1/\varepsilon_w$ [10].* For our test cases, we found $\kappa_{\text{norm}} < 1/\gamma\varepsilon_w$ small enough to provide reliable normwise

*A more precise but more complicated bound appears in [13].

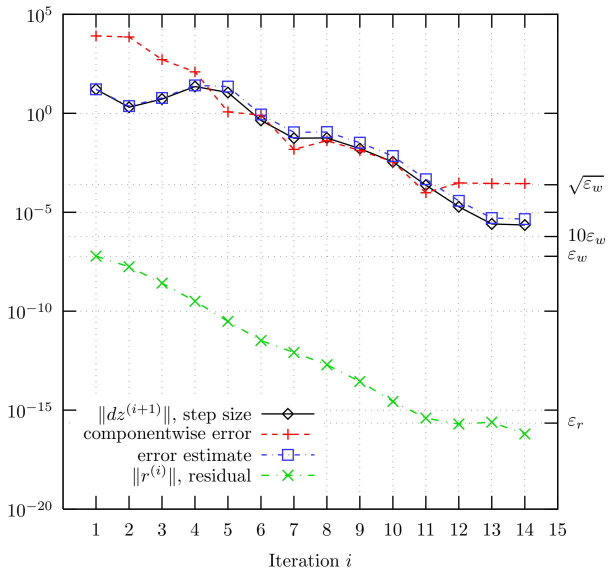


Figure 16: Componentwise error is *under*estimated by a factor of over 80 for a very ill-conditioned system, $\kappa_{\text{comp}} \approx 4.2 \times 10^{12}$. Here $E_{\text{comp}} \approx 2.81 \times 10^{-4}$, $B_{\text{comp}} \approx 3.20 \times 10^{-6}$. Throughout this section, $\rho_{\text{thresh}} = 0.5$, $\varepsilon_w \approx 5.96 \times 10^{-8}$, $\gamma\varepsilon_w = 10\varepsilon_w \approx 5.96 \times 10^{-7}$, $\sqrt{\varepsilon_w} \approx 2.44 \times 10^{-4}$ and $\varepsilon_r \approx 2.22 \times 10^{-16}$. The step size and error estimate are almost equal and mostly overlap.

results and error estimates. The threshold $1/\gamma\varepsilon_w$ is approximately 1.7×10^6 for our 100×100 test cases. The componentwise results are reliable when $\kappa_{\text{comp}} < 1/\gamma\varepsilon_w$.

If a user requests solution of an extremely ill-conditioned matrix, our bounds can under- or overestimate the error severely. When faced with Rump's outrageously ill-conditioned matrices [24] and random x, our algorithm either successfully solved the systems ($O(\varepsilon_w)$ errors and bounds) or correctly reported failure. However, Table 3 shows a small number of *extreme* mis-estimates, those off by more than 100×, occur with normwise and componentwise ill-conditioned systems.

The extreme *under*estimates occur when refinement converges well to a "wrong" solution. Figure 16 shows the iteration history of one such matrix from Section 5.2's test suite.* This matrix has $\kappa_{\text{comp}} \approx 4.2 \times 10^{12}$, far above the $1/\gamma\varepsilon_w \approx 1.7 \times 10^6$ threshold. Algorithm 3 finds $B_{\text{comp}} \approx 3.20 \times 10^{-6}$ for a true error $E_{\text{comp}} \approx 2.81 \times 10^{-4}$, underestimating the componentwise error by over a factor of 80. Refinement terminates for lack of progress. The final computed $\hat{x}$ has a tiny residual, $\|\hat{r}\| < \varepsilon_r \approx 10^{-16}$. As far as the residual is concerned, the computed $\hat{x}$ solves $Ax = b$ as well as the true solution x does. These underestimates are unsurprising and unavoidable with finite precision refinement.

The extreme *over*estimates may occur for two reasons, early termination and step magnification. When Algorithm 3 terminates because $i = i_{\text{thresh}}$, the current $dx^{(i)}$ is still improving the solution. The final $\|dx^{(i)}\|/\|x^{(i)}\|$ may be large although the error is small. Well-conditioned systems converge almost immediately, so this can occur only for ill-conditioned systems.

Figure 17 illustrates the second cause of overestimates, steps magnified by an ill-conditioned A. Here a small residual is magnified at iteration 4 to a large step, triggering the no-progress termination criterion (17). Six further iterations show that the error has stopped decreasing, and

*This underestimate can be reproduced with `./driver -u 0.5 -n 100 -seed 1972 97 1383 1741`.

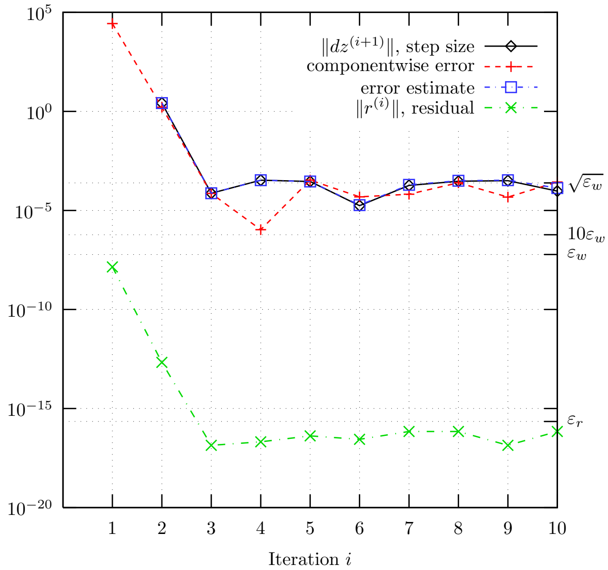


Figure 17: Componentwise error is *over*estimated at iteration 4 by over a factor of 100. The system is very ill-conditioned, $\kappa_{\text{comp}} \approx 4.2 \times 10^{12}$. Here $E_{\text{comp}}^{(4)} \approx 1.06 \times 10^{-6}$ and $B_{\text{comp}}^{(4)} \approx 3.37 \times 10^{-4}$. The step size and error estimate are almost equal and mostly overlap.

that our estimate would have matched the error had the iteration continued. This matrix is ill-conditioned and ill-scaled*, with $\kappa_{\text{comp}} \approx 1.6 \times 10^{13}$ and with a single column scaled by 2^{17}. The residual drops below ε_r while the step $\|dz\|$ and bound B_{comp} hover around 10^{-4}. Algorithm 3 terminates for lack of progress at iteration 4. The delivered solution has true componentwise error $E_{\text{comp}} \approx 10^{-6}$ and overestimates that error by two orders of magnitude. Again, this is unsurprising and unavoidable with ill-conditioned matrices and finite-precision arithmetic.

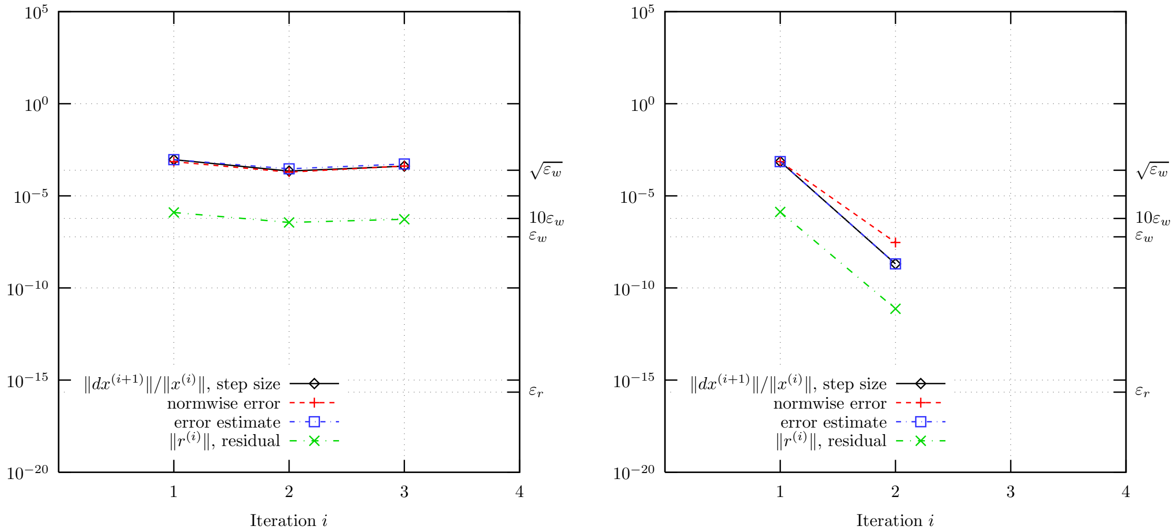
It is also possible that the LU factorization of an ill-conditioned A is so poor that we solve entirely the wrong system and underestimate our true error. In general, we cannot expect factorization to identify all singular matrices. When presented with a singular system, however, Algorithm 3 computes large estimates B_{norm} and B_{comp}. To identify potentially singular matrices, any estimate at least $\sqrt{\varepsilon_w}$ is considered "infinite" and is set to one.

Consider Example 2.6 from [10], modified for single-precision IEEE754 arithmetic. The example involves the exactly singular matrix

$$A = \begin{bmatrix} 3 \cdot 2^{-7} & -2^{7} & 2^{7} \\ 2^{-7} & 2^{-7} & 0 \\ 2^{-7} & -3 \cdot 2^{-7} & 2^{-7} \end{bmatrix}.$$

If we store $b = A \cdot [1, 1 + \varepsilon_w, 1]^T$ as single-precision data, we introduce enough error to ensure that $Ax = b$ has *no* single-precision solution. Factorization succeeds in single precision without equilibration, and subsequent refinement estimates a normwise relative $B_{\text{norm}} \approx 0.3$. Because there is no solution, the true normwise relative error is infinite. When factored and solved in double precision, refinement computes $B_{\text{norm}} \approx 10^{14}$. With Section 2.2's equilibration, this particularly simple matrix is correctly identified as singular during factorization.

*Produced by `./driver -u 0.5 -n 100 -seed 1235 3091 2150 2005` .



(a) Residual calculated in single precision

(b) Residual calculated in double precision

Figure 18: Calculating the residual in single precision prevents the normwise error from reaching $\gamma\varepsilon_w$. Here $\kappa_{\text{norm}} \approx 1.9 \times 10^4$. The step size and error estimate are almost equal and mostly overlap.

Because these effects are unavoidable, cautious users should declare B_{norm} and B_{comp} unreliable when $\kappa_{\text{norm}} \geq 1/\gamma\varepsilon_w$ (normwise results) or $\kappa_{\text{comp}} \geq 1/\gamma\varepsilon_w$ (componentwise). As an additional precaution, our implementation treats B_{norm} or $B_{\text{comp}} \geq \sqrt{\varepsilon_w}$ as a failure to converge. Any error bound which does not converge is set to one. These thresholds are not guarantees, but we have neither encountered nor constructed systems which pass these tests yet defeat our estimators.

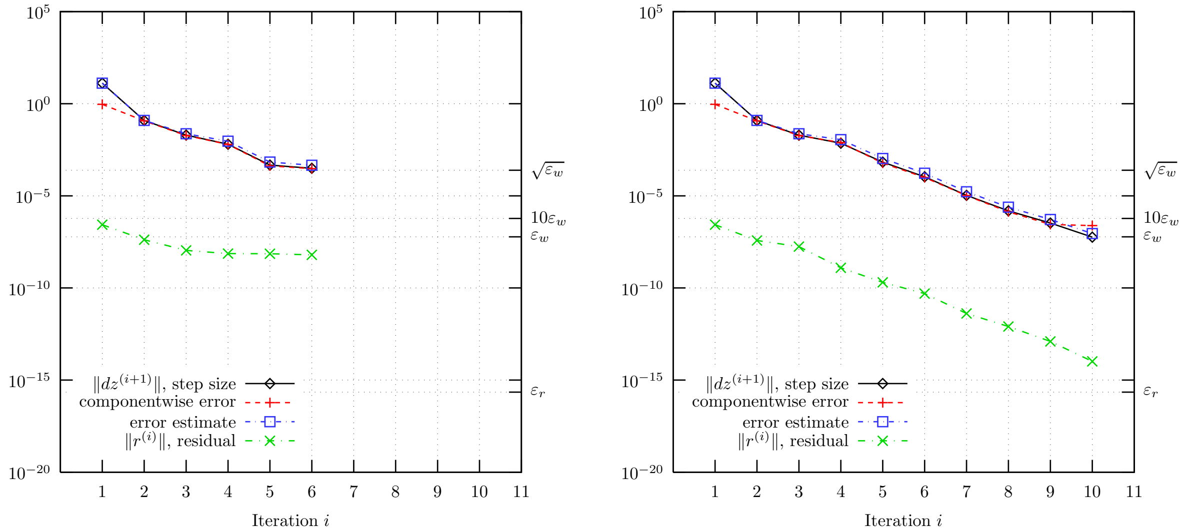
7.2 Rounding Errors in Residual and Update Computations

The rounding errors in the residual (δr) and update (δy) were assumed negligible before termination in Section 2. When paired with ill-conditioning, however, these errors prevent convergence and can cause significant underestimates. Ultimately, round-off errors in the residual do not matter except for extremely ill-conditioned systems. Round-off in the update, however, requires special handling to achieve a reliable componentwise error estimate.

Algorithm 3's error estimates and termination criteria do not directly include the residual. Rounding in the residual is magnified through a condition number often close to our κ_{norm} [13] and impacts only the computation of $dy^{(i+1)}$. Figure 18 shows how single-precision residuals prevent refinement from reducing even the normwise error for ill-conditioned systems ($\kappa_{\text{norm}} \approx 1.9 \times 10^4$).* The ratio of the error's norm to the residual's norm is roughly constant around 1000 for both the single- and double-precision calculations. Because the single-precision residual's error is limited by $\varepsilon_w \approx 10^{-8}$, the error will not decrease below $10^{-5} > \gamma\varepsilon_w$.

If the residual is tiny and the system is well-conditioned by any reasonable measure, then the computed $y^{(i)}$ is a good approximation to y. Underestimating any error is unlikely. At worst, the round-off could increase $dy^{(i+1)}$ normwise or though some scaling, producing an overestimate

*Produced by `./driver -u 0.5 -n 100 -seed 754 4072 1172 2893`, with the both residual and solution limited to single precision by `-precs 0 0`.



(a) Solution in single precision

(b) Solution in doubled precision

Figure 19: Here refinement converges componentwise by carrying the computed solution to doubled precision. The step size and error estimate are almost equal and mostly overlap.

of the true error. If the system is ill-conditioned by our κ_{norm} or κ_{comp} measure, we have already dismissed the results' reliability. Condition numbers near our threshold $1/\gamma\varepsilon_w$ *may* encounter under- or over-estimates. We have neither encountered nor successfully constructed such cases.

The rounding errors $\delta y^{(i)}$ from updating $y^{(i)} = y^{(i-1)} + dy^{(i)} + \delta y^{(i)}$ also limit our accuracy. Following errors from Equation (5) through Equations (3) and (4), we see that the effect of $\delta y^{(i)}$ on the *next* update $dy^{(i+1)}$ is $(A_s + \delta A_s^{(i+1)})^{-1} A_s \delta y^{(i)}$. The magnitude of $\delta y^{(i)}$ is bounded by $\varepsilon_x |y^{(i)}|$, and we expect this error to affect the normwise iteration only when $\kappa_s \varepsilon_x = \kappa_\infty(A_s)\varepsilon_x$ significantly exceeds ε_w.

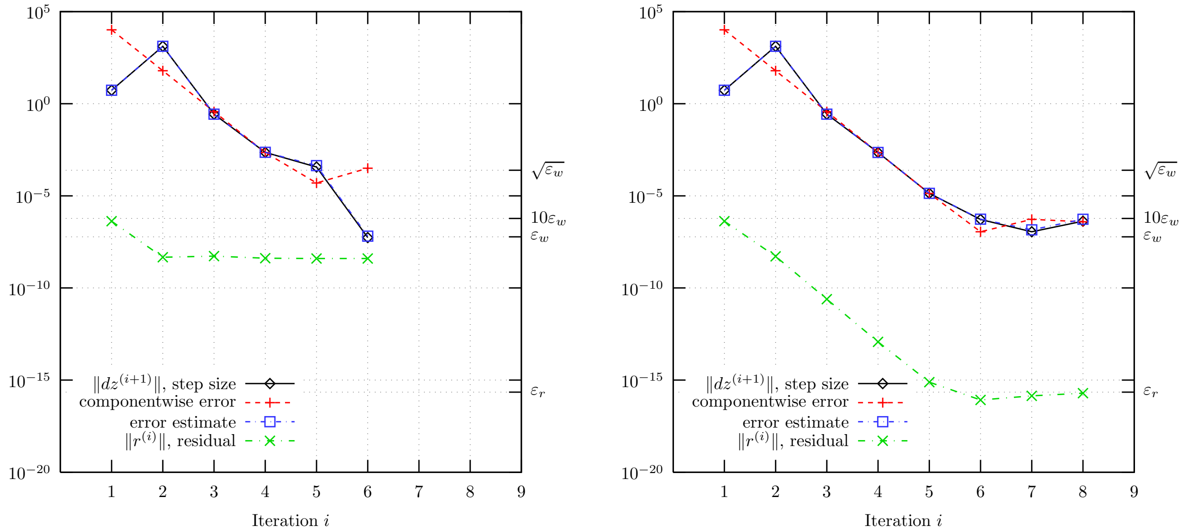
The effect on the componentwise iteration, however, depends on $dz^{(i+1)} = C_z dy^{(i+1)}$, where $C_z = \operatorname{diag}(y)^{-1}$. If we scale $(A_s + \delta A_s^{(i+1)})^{-1} A_s \delta y^{(i)}$ by C_z and assume that $(A_s + \delta A_s^{(i+1)})^{-1} \approx A_s^{-1}$, then the effect on step $i+1$'s update of rounding during step i's update is bounded normwise by

$$\| \operatorname{diag}(y)^{-1} \cdot (A_s + \delta A_s^{(i+1)})^{-1} A_s \cdot \operatorname{diag}(y)\, \delta z^{(i)} \| \quad \lessapprox \quad \varepsilon_x \, \|(A_s \operatorname{diag}(y))^{-1}\| \, \|A_s \operatorname{diag}(y)\|.$$

We approximate the right-hand quantity with the computed solution $\hat{y}$ in our κ_{comp}. Representation error ε_x only affects the componentwise solution when κ_{comp} is sufficiently large.

Our algorithm squares ε_x whenever it suspects $\kappa_{\text{comp}} \geq 1/\gamma\varepsilon_w$, so the update's effect remains negligible. The iteration history in Figure 19 shows how the doubled precision allows the residual's continued decrease.* In this example, $\kappa_{\text{comp}} \approx 1.2 \times 10^8$, and Algorithm 3 switched to carrying $y^{(i)}$ to doubled precision after the first iteration. Both componentwise and normwise solutions halted at iteration 6 with single-precision updates, resulting in errors of around 10^{-4}. Refinement with double-precision updates continues for ten iterations and reduces the error below $\gamma\varepsilon_w$. The crucial rounding errors occur moving from iteration 3 to iteration 4. The difference in magnitude between

*Produced by `./driver -u 0.5 -n 100 -seed 3346 3503 2135 1313`, with solution precision limited to single by `-precs 1 1`.



(a) Solution in single precision

(b) Solution in doubled precision

Figure 20: Carrying the solution to doubled precision prevents severe componentwise error underestimates. This refinement would underestimate the componentwise error by almost a factor of 5000. The step size and error estimate are almost equal and mostly overlap.

single-precision y and doubled precision for the relative step $\|dx^{(3)}\|/\|x^{(3)}\|$ is around $2.8 \times 10^{-5} \approx 470\varepsilon_w$. The difference between values $\|dz^{(3)}\|$ is around $3.8 \times 10^{-5} \approx 640\varepsilon_w$. Those small differences in the updates allow Algorithm 3 to achieve componentwise and normwise errors of 10^{-7}.

Without this extra check, the computed solution potentially could have large componentwise error which would be underestimated drastically. Figure 20 shows one such underestimate,* with $\kappa_{\text{comp}} \approx 1.3 \times 10^{11}$ and $E_{\text{comp}} \approx 5000 B_{\text{comp}}$. There is one component tiny in both x and $y = Cx$, and only that component still is unconverged componentwise by the fourth iteration. But single-precision rounding errors in updates halted the residual's norm decrease by the third iteration. The resulting $dz^{(i)}$ steps become similar to noise, eventually rounding the wrong way and "accidentally" converging.

The most effective solution we have found is to increase the solution's precision, decreasing ε_x and preventing update rounding errors from compounding as quickly. Carrying $y^{(i)}$ to twice the working precision achieves componentwise accuracy and reliable estimates in our tests. The residual is computed with the full $y^{(i)}$, but the step $dy^{(i+1)}$ still is computed only to ε_w. But running with $\varepsilon_x \leq \varepsilon_w^2$ is expensive; such an iteration on our test platform takes around 1.5× as long to compute. Using a doubled precision [19], we dynamically extend ε_x from ε_w to ε_w^2 when $\max_k y_k^{(i)}/\min_k y_k^{(i)} \geq 1/\gamma\varepsilon_w$. Dynamically increasing precision reduces the worst normwise underestimate factor from 1010 to 230 and the worst componentwise underestimate factor from 6300 to 320. An alternative is to modify $\rho_{\text{max,z}}$ by using $\max\{dy^{(i)}, \varepsilon_x \|y^{(i)}\|_\infty\}$ as the denominator. This alternative avoids the underestimates, but it also weakens the error estimate for many well-behaved cases and does not improve the true componentwise error.

*Produced by `./driver -u 0.5 -n 100 -seed 3326 1514 1218 4009`, with solution precision limited to single by `-precs 1 1`.

7.3 Zero Components and Scaling

True solutions to linear systems may have exact zero components. These appear in optimization applications when solving for directional derivatives at optimal or saddle points, in physical models where forces or currents are balanced, *etc.* Exact or near zero entries could induce division by zero or overflow when calculating the componentwise change dz. Our implementation protects against zero components but does not use a threshold for tiny components. Exceptionally large entries in the solution could cause underflows, but that underflow is correct; tiny dz components will not change their solution entries.

First consider exact zero solutions from $Ax = 0$. If factorization of A_s succeeds, Algorithm 3 calculates $y^{(1)} = 0$ exactly. The first residual $r^{(1)} = 0$, so the step $dy^{(2)} = 0$. When calculating $\|dx^{(i+1)}\|/x^{(i+1)}$ and $\|dz^{(i+1)}\|$, our implementation tests for zeros and substitutes the result $0/0 = 0$. So both $B_{\text{norm}} = 0$ and $B_{\text{comp}} = 0$. These tests also ensure our implementation does not encounter IEEE754 exceptional behavior unnecessarily. Our implementation also ensures that systems with block structure like

$$\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ 0 \end{bmatrix}$$

return the same bounds as the system $A_1 x_1 = b_1$. Also, any component, zero or not, occurring during refinement is considered exact so long as the corresponding component of dx is also zero. Rounding errors that result in computing $dx_k^{(i+1)} = 0$ could lead to falsely declaring the k-th component "exact." We do not protect our purely relative error bounds against these accidentally-zero dx components; most such protections report large error bounds when solving $Ax = 0$.

When our implementation encounters a zero solution component corresponding to a non-zero component of $dy^{(i+1)}$, that step's norm $\|dz^{(i+1)}\|$ is set to a huge value. The componentwise solution then is declared unstable (line 5 in Procedure new-z-state), and $\mathsf{final\text{-}relnorm}_z$ is set to ∞. If the normwise solution has converged, refinement terminates and reports $B_{\text{comp}} = \infty$. This behavior is correct and cautious; the component may not be zero and we do not know even its sign. The componentwise solution may re-stabilize once it has passed through the zero.

Dividing by a tiny solution component could cause an overflow while calculating $\|dz^{(i)}\|$. In this case, the corresponding component of $|dy^{(i)}|$ is greater than the component of $|y^{(i)}|$. We assume that component's sign is not specified accurately; the overflow yields a correctly large componentwise relative error.

We have tested exact zero solution components with a special generator included with our research code. The Octave [12] code (MATLAB$^{\text{TM}}$-compatible [20]) for this generator is in Appendix A. We have neither encountered nor constructed tests where the calculation of $\|dz^{(i)}\|$ induces an overflow, but the code handles infinite $\|dz^{(i)}\|$ correctly. Such a $\|dz^{(i)}\|$ will send the componentwise solution back to the unstable state; see Procedure new-z-state on page 15.

7.4 Equilibration

As discussed in Section 2.2, we equilibrate the input system to ameliorate the effects of scaling on a system's conditioning. Our algorithm handles many ill-scaled systems well. Section 2.2's equilibration fixes matched ill-scalings, where an ill-scaled column (or row) corresponds to a similarly ill-scaled component in x (or b).

Consider a system $A_s y = b_s$ that is not too ill-conditioned on its own and a very ill-conditioned, diagonal R. If the rows and right-hand side are scaled by $RA_s y = Rb_s$, the resulting system will have a large κ_∞ but small κ_{norm}. Unless the scaling loses information through over- or underflow, simple equilibration allows refinement to produce an accurate answer to the ill-conditioned system.

Similar column and solution scaling by $(A_s C^{-1}) \cdot (Cy) = b$ introduces one limitation, however. The infinity norm of $x = Cy$ may be dominated by a single component; consider $C = \text{diag}(10^7, 1, 1, \ldots)$. If that single component converges quickly enough, the componentwise changes will not have stabilized, and Algorithm 3 will not produce a componentwise accurate answer. The smaller components can be completely wrong! These drastic scalings produce large κ_{comp} condition numbers; Section 5.2's test suite includes some of these cases in the componentwise ill-conditioned results.

Section 2.2's equilibration does not fix all cases of ill-scaling. Consider the ill-scaled matrix

$$\begin{bmatrix} 0 & G & G \\ G & g & 0 \\ G & 0 & g \end{bmatrix},$$

where G is extremely large and g extremely small. Our equilibration retains this matrix's ill-scaling and ill-conditioning.

In the most extreme case, G is the overflow threshold and g is the underflow threshold. Our equilibration reduces this matrix to the singular matrix

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

Moreover, LAPACK's current refinement routine equilibrates the input matrix *in-place*, overwriting the user's matrix. Underflow in equilibration destroys the user's copy and unpleasantly affects many subsequent computations. Fixing this problem requires changes to LAPACK which currently are under consideration.

8 New Routines Proposed for LAPACK

The current LAPACK driver routine `xGESVX` calls the working precision iterative refinement routine `xGERFS` which implements a variant of Algorithm 5. We propose to enhance LAPACK with the new routines `xGESVXX/xGERFSX`, which include both Algorithms 3 and 5. It is worth pointing out that the amount of work space for the new routines has not changed.

The following is the calling sequence of the current LAPACK routine `SGERFS` with single precision iterative refinement:

```
      SUBROUTINE SGERFS( TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB,
     +                   X, LDX, FERR, BERR, WORK, IWORK, INFO )
```

The new routine `SGERFSX` has the following calling sequence:

```
      SUBROUTINE SGERFSX( TRANS, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV,
     +                    R, C, B, LDB, X, LDX, RCOND, FERR, FERR_CMP,
     +                    BERR, NPARAMS, PARAMS, WORK, IWORK, INFO )
```

The new arguments are EQUED, R, C, RCOND, FERR_CMP, and an array of optional parameters of length NPARAMS stored in PARAMS. EQUED specifies the form of equilibration performed on A before calling this routine. If A was equilibrated, R and C contain the row and column scale factors. RCOND, the condition number $\kappa_\infty(A_s)$, has been added as an input argument; it is used when deciding to carry y (stored in X) to extra precision. The new output argument is FERR_CMP, which returns the componentwise error bound (B_{comp} in Algorithm 3) for each right-hand side. There is one notable change in FERR and RCOND, as well. All quantities are now based on the stricter ∞-norm instead of the 1-norm. Also, before returning to the user, any FERR or FERR_CMP at least $\sqrt{\varepsilon_w}$ is set to one.

The argument array PARAMS of length NPARAMS holds optional parameters. The symbolic names are available through an include file*. Only parameters from 1 to NPARAMS are referenced; if NPARAMS ≤ 0, PARAMS is not referenced and defaults are used. If a parameter entry passed to the routine is negative, that parameter is replaced on output by the value used in the routine.

The following parameters are passed in PARAMS:

PARAMS(ITREF_PARAM = 1) Precision used in performing iterative refinement. Symbolic names are defined by the XBLAS. The default for single precision is BLAS_PREC_DOUBLE; defaults for other precisions have not been determined. The following description is for the single-precision SGERFSX code. See [17] for how BLAS_PREC_* affects other precisions.

- 0 Do not perform refinement.
- BLAS_PREC_SINGLE Perform single-precision refinement. The routine is similar to the current LAPACK routine SGERFS, with the following modifications (see Algorithm 5):
  - the column scaling factor C (or the row scaling factor R for the transposed system) is directly applied to the scaled solution when estimating the error bound FERR, which gives a better estimate for the solution of the original system, and
  - a componentwise error bound FERR_CMP is computed and returned.
- BLAS_PREC_DOUBLE Perform the double-precision refinement as specified in Algorithm 3.
- BLAS_PREC_INDIGENOUS For SGERFSX, if the compilation environment supports at least double precision, act as if BLAS_PREC_DOUBLE. Otherwise act as if BLAS_PREC_SINGLE.
- BLAS_PREC_EXTRA Use intermediate precision at least 1.5 times the base precision. In SGERFSX, this is effectively BLAS_PREC_DOUBLE.

PARAMS(CONDTHRESH_PARAM = 2) Condition number threshold where the error estimates are no longer considered trustworthy. Change with extreme caution. Defaults to $\gamma\varepsilon_w$.

PARAMS(ITHRESH_PARAM = 3) Total number of residual computations allowed for refinement. Defaults to 10 for double-precision refinement, 5 for single-precision refinement. Set this to 100 for our "aggressive" settings.

PARAMS(COMPONENTWISE_PARAM = 4) Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Defaults to 1.0.

*The *_PARAM names and definitions likely will be changed once included into LAPACK proper. They are included here for discussion.

PARAMS(RTHRESH_PARAM = 5) Our ρ_{thresh} used in criterion (17), the ratio of consecutive "step sizes" required to continue relative normwise or componentwise refinement. Defaults to 0.5. Set this to 0.9 for our "aggressive" settings.

PARAMS(DZTHRESH_PARAM = 6) Our dz_{thresh}, the threshold where the solution is considered stable enough for computing componentwise measurements. Defaults to 0.25.

The following is the calling sequence of the new driver routine SGESVXX:

```
      SUBROUTINE SGESVXX( FACT, TRANS, N, NRHS, A, LDA, AF, LDAF,
     +                    IPIV, EQUED, R, C, B, LDB, X, LDX, RCOND,
     +                    FERR, RCOND_NRM, FERR_CMP, RCOND_CMP, BERR,
     +                    NPARAMS, PARAMS, WORK, IWORK, INFO )
```

Compared with the current driver routine SGESVX, the new arguments are FERR_CMP (output), RCOND_NRM (output), RCOND_CMP (output), and the parameter array PARAMS (input / output) of length NPARAMS (input). FERR_CMP, NPARAMS, and PARAMS are as explained above. RCOND_NRM returns κ_{norm} for the entire system, and the NRHS-long array RCOND_CMP returns κ_{comp} for each right-hand side. A cautious user should disregard FERR if RCOND_NRM $< \gamma\varepsilon_w$, and also FERR_CMP if RCOND_CMP $< \gamma\varepsilon_w$. The value $\gamma\varepsilon_w$ suggested for given parameters is returned to the user in PARAMS(CONDTHRESH_PARAM) when a negative number is passed in that parameter, but $\gamma\varepsilon_w$ is always safe. As with SGERFSX, the estimates and condition numbers are now based on the ∞-norm, and estimates at least $\sqrt{\varepsilon_w}$ are set to one.

9 Conclusions and Future Work

We have presented a new variation on the extra precise iterative refinement algorithm for the solution of linear systems of equations. With negligible extra work we can return a bound on the maximum relative error in any solution component, as well as the traditional normwise error bound. We prove this by means of an error analysis that exploits the column scaling invariance of the algorithm. With the availability of the extended precision BLAS standard, the algorithm can be implemented in a portable way. Based on a large number of numerical experiments (two million each of 5×5, 10×10, and 100×100 test matrices, and two hundred thousand 1000×1000 matrices), we show that the algorithm converges quickly for all but the worst conditioned problems (i.e. for condition numbers no larger than about the reciprocal of machine precision $1/\varepsilon_w$) and that the corresponding error bounds are very reliable. The algorithm also converges for a large fraction of the extremely ill-conditioned problems (with condition numbers exceeding $1/\varepsilon_w$) although the error bounds occasionally underestimate the true error. Some difficulties with the badly scaled problems (i.e. with greatly varying solution components) can be overcome by using extra precision for the updated x (the so-called double-x iteration).

In particular, as long as a normwise condition number κ_{norm} computed by the algorithm does not exceed $1/\gamma\varepsilon_w$, the algorithm returned a tiny, correct normwise error bound in *all* cases tested. Similarly, as long a componentwise condition number κ_{comp} computed by the algorithm does not exceed $1/\gamma\varepsilon_w$, the algorithm returned a tiny, correct componentwise error bound in all cases tested. Based on these results, we believe the algorithm is very reliable.

Programming systems like MATLAB [20] that are used to solve $Ax = b$ may return a warning that A is nearly singular, based on a condition estimator. This condition estimator, like our algorithm, costs just a few triangular solves, i.e. very little extra beyond the triangular factorization for medium to large n. Therefore, these programming systems could consider using iterative refinement as a default, issuing a warning only if the system is not guaranteed to be fully accurate, because κ_{norm} (or κ_{comp}) is too large.

Our algorithm applies to all the other LAPACK [1] and ScaLAPACK [4] linear system solvers. Additional structure in symmetric and banded systems may allow better error estimates or earlier termination. Section 2's error analysis needs to be extended to these systems. Choosing appropriate condition numbers for symmetric linear systems presents an interesting challenge. Our $\kappa_{\text{norm}} = \kappa_\infty(R \cdot A)$ assumes that the domain and range can be scaled independently, but the two are scaled simultaneously for symmetric systems.

The majority of computers contain processors implementing Intel's IA32 architecture [15]. These computers support 80-bit floating-point arithmetic at full hardware speed. The 80-bit arithmetic is an implementation of IEEE754 double-extended precision. Future work will extend Algorithm 3 and its error analysis to include using this kind of extended precision.

We also plan to study how aspects of Algorithm 3 benefit sparse linear systems. In particular, these techniques may assist our parallel sparse direct solver SuperLU_DIST [18], where static pivoting instead of partial pivoting is used for numerical stability. In an unpublished report [36], Wilkinson points out this potential benefit even with single precision residual computations: "... when x^0 has been determined by a direct method of some *poorer* numerical stability than Gaussian elimination with pivoting ... the use of $d^{(0)}$ as an actual correction should yield substantial dividends ... and may be of great value in the solution of sparse systems when pivoting requirements have been relaxed." Our additional contributions, improved termination criteria and additional precision for the solution, may carry refinement even further.

A Generating Systems with Exact Zero Solution Components

The following Octave [12] (or MATLAB$^{\text{TM}}$ [20]) function generates an ill-conditioned test system $Ax = b$ where the returned solution is exact and can contain exact zero components.

```
function [A, X, B] = crnd(nn, kk, p)
% [A,X,B] = crnd(n,k,p) returns three n x n matrices among which
% A/p and p*X have random integer entries, and B = (A/p)*(p*X)
% exactly, but X has mostly noninteger entries. A can be ill-
% conditioned too; its condition number is at least about 2^k .
% If omitted, k defaults to 16; otherwise 2=< k =< 18. And
% 2 < |p| = (a small odd integer, preferably a prime) < 16; its
% default is 3. If p > 0 then some entries of X will be zeros.
if nargin < 3, p = 3; end
sp = (p > 0); p = abs(p);
if not(any([3 5 7 9 11 13 15] == p)), sp = 1;
   disp('p in crnd(n,k,p) has been changed to'), p = 3 , end
if nargin < 2, kk = 16; end
k = max(2, min(18, round(kk))); if (k ~= kk),
      disp('k in crnd(n,k,p) has been changed to'), k, end
n = max(3, min(1000, round(nn))); if (n ~= nn),
```

```
    disp('n in crnd(n,k,p) has been changed to'), n , end
  tk = 2^k ;  % ... condition no. of A will be at least about tk .
  A = fix( (2*tk)*(rand(n) - 0.5) ) ; %... random k-bit integers.
  X = inv(A) ;  [r, i] = max(abs(X)) ;  [c, j] = max(r) ;  i = i(j) ;
  A(j,i) = A(j,i) - round( 1/X(i,j) ) ;% ... makes A ill-conditioned.
  c = (2^( 23 - k ))/n ;  % ... since arithmetic carries 24 sig, bits.
  X = fix( (2*c)*rand(n) - c ) ; %... random (52-k)-bit integers.
  while sp&any(all(X)),  % ... sprinkle some zeros into X :
      r = 0 ;
      while (any(all(r==0))),  r = (rand(n) > 0.25) ;
      while any(all(r)),  r = (rand(n) > 0.25).*r ;  end, end
      X = r.*X ;
    end % ... sprinkling zeros
  B = A*X ; %... exactly, with integer entries at most 23 bits wide.
  A = p*A ; %... exactly, with integer entries at most k+4 bits wide.
  X = X/p ; %... rounded, with rounding errors each at worst 1/2 ulp.
```

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide, Release 3.0.* SIAM, Philadelphia, 1999. URL `http://www.netlib.org/lapack/lug/`. 407 pages.

[2] *IEEE Standard for Binary Floating Point Arithmetic.* ANSI/IEEE, New York, Std 754-1985 edition, 1985. URL `http://grouper.ieee.org/groups/754/`.

[3] Åke Björck. Iterative refinement and reliable computing. In M.G. Cox and S.J. Hammarling, editors, *Reliable Numerical Computation*, pages 249–266. Oxford University Press, 1990.

[4] L. S. Blackford, J. Choi, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide.* SIAM, Philadelphia, 1997. URL `http://www.netlib.org/scalapack/slug/`. 325 pages.

[5] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, Z. Maany, F. Krough, G. Corliss, C. Hu, B. Keafott, W. Walster, and J. Wolff v. Gudenberg. Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard. *Intern. J. High Performance Comput.*, 15(3-4), 2001. URL `http://www.netlib.org/blas/blast-forum/`.

[6] H.J. Bowdler, R.S. Martin, G. Peters, and J.H. Wilkinson. Handbook series linear algebra: Solution of real and complex systems of linear equations. *Numerische Mathematik*, 8:217–234, 1966.

[7] Yang Cao and Linda Petzold. A subspace error estimate for linear systems. *SIAM Journal on Matrix Analysis and Applications*, 24(3):787–801, 2003. URL `http://epubs.siam.org/sam-bin/dbq/article/39064`.

[8] S. Chandrasekaran and I. C. F. Ipsen. On the sensitivity of solution components in linear systems of equations. *SIAM Journal on Matrix Analysis and Applications*, 16(1):93–112, 1995. URL http://epubs.siam.org/sam-bin/dbq/article/23125.

[9] Intel Corporation. Math kernel library 7.2. URL http://www.intel.com/software/products/mkl/.

[10] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[11] J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, 1979.

[12] John W. Eaton. *GNU Octave Manual*. Network Theory Limited, 2002. ISBN 0-9541617-2-6. URL http://www.octave.org/.

[13] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, 2002. ISBN 0-89871-521-0. URL http://www.ma.man.ac.uk/~higham/asna/.

[14] American National Standards Institute. *American National Standard programming language, FORTRAN*. American National Standard; ANSI X3.9-1978 CSA standard; Z243.18-1980 American National Standards Institute. American National Standard; ANSI X3.9-1978. Canadian Standard Association. CSA standard; Z243.18-1980. American National Standards Institute, 1430 Broadway, New York, NY 10018, USA, revised edition, 1978. URL http://www.fortran.com/fortran/F77_std/rjcnf.html.

[15] *IA-32 Intel^TM Architecture Software Developer's Manual, Volume 1: Basic Architecture*. Intel Corporation, 2004. URL http://developer.intel.com/design/pentium4/manuals/index_new.htm#sdm_vol1. Order #253665.

[16] Andrzej Kiełbasiński. Iterative refinement for linear systems in variable-precision arithmetic. *BIT*, 21:97–103, 1981.

[17] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Y. Kang, A. Kapur, M. C. Martin, B. J. Thompson, T. Tung, and D. J. Yoo. Design, Implementation and Testing of Extended and Mixed Precision BLAS. *ACM Transactions on Mathematical Software*, 28(2):152–205, 2002. URL http://www.nersc.gov/~xiaoye/XBLAS.

[18] Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2):110–140, June 2003. URL http://doi.acm.org/10.1145/779359.779361.

[19] Seppo Linnainmaa. Software for doubled-precision floating-point computations. *ACM Transactions on Mathematical Software*, 7(3):272–283, September 1981. ISSN 0098-3500. URL http://doi.acm.org/10.1145/355958.355960.

[20] MathWorks, Inc. Matlab^TM. URL http://www.mathworks.com/.

[21] Sun Microprocessors. Performance libraries 6.0. URL http://developers.sun.com/prodtech/cc/perflib_index.html.

[22] Cleve B. Moler. Iterative refinement in floating point. *Journal of the Association for Computing Machinery*, 14(2):316–321, 1967. URL `http://doi.acm.org/10.1145/321386.321394`.

[23] UML 1.4. *Unified Modelling Language Specification, version 1.4*. Object Modeling Group, September 2001. URL `http://www.omg.org/cgi-bin/doc?formal/01-09-67`.

[24] Siegfried M. Rump. A class of arbitrarily ill conditioned floating-point matrices. *SIAM Journal on Matrix Analysis and Applications*, 12(4):645–653, October 1991. URL `http://locus.siam.org/SIMAX/volume-12/art_0612049.html`.

[25] Siegfried M. Rump. Structured perturbations part I: Normwise distances. *SIAM Journal on Matrix Analysis and Applications*, 25(1):1–30, January 2004. ISSN 0895-4798 (print), 1095-7162 (electronic). URL `http://epubs.siam.org/sam-bin/dbq/article/40573`.

[26] Siegfried M. Rump. Structured perturbations part II: Componentwise distances. *SIAM Journal on Matrix Analysis and Applications*, 25(1):31–56, January 2004. ISSN 0895-4798 (print), 1095-7162 (electronic). URL `http://epubs.siam.org/sam-bin/dbq/article/40574`.

[27] S.M. Rump. Solving algebraic problems with high accuracy. In U.W. Kulisch and W.L. Miranker, editors, *A New Approach to Scientific Computation*, pages 51–120. Academic Press, 1983.

[28] S.M. Rump. Verified computation of the solution of large sparse linear systems. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 75:S439–S442, 1995.

[29] R. D. Skeel. Iterative refinement implies numerical stability for Gaussian elimination. *Math. Comput.*, 35:817–832, 1980.

[30] G. W. Stewart. *Introduction to Matrix Computations*. Academic Press, New York, 1973. ISBN 0-89871-355-2. xiii+441 pp.

[31] V. Strassen. Gaussian Elimination is not optimal. *Numerical Mathematica*, 13:354–356, 1969.

[32] L.N. Trefethen and R.S. Schreiber. Average-case stability of gaussian elimination. *SIAM Journal on Matrix Analysis and Applications*, 11(3):335–360, 1990. URL `http://locus.siam.org/SIMAX/volume-11/art_0611023.html`.

[33] A. van der Sluis. Stability of solutions of linear algebraic systems. *Numerische Mathematik*, 14:246–251, 1970.

[34] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001. URL `http://www.netlib.org/lapack/lawns/lawn147.ps`. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (`http://www.netlib.org/lapack/lawns/lawn147.ps`).

[35] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963. ISBN 0-486-67999-3. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New York, 1994.

[36] J.H. Wilkinson. The use of single-precision residuals in the solution of linear systems. Unpublished manuscript, NPL, 1977.