Error Bounds from Extra-Precise Iterative Refinement

JAMES DEMMEL, YOZO HIDA, and WILLIAM KAHAN
University of California, Berkeley
XIAOYE S. LI
Lawrence Berkeley National Laboratory
SONIL MUKHERJEE
Oracle
and
E. JASON RIEDY
University of California, Berkeley

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 1948 and analyzed in the 1960s as a means to compute very accurate solutions to all but the most ill-conditioned linear systems. 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 has essentially removed the first obstacle. To overcome the second obstacle, we show how the 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.

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.
Authors' addresses: J. Demmel, Computer Science Division and Mathematics Dept., University of California, Berkeley, CA 94720; email: demmel@cs.berkeley.edu; Y. Hida, Computer Science Division, University of California, Berkeley, CA 94720; email: yozo@cs.berkeley.edu; W. Kahan, Computer Science Division and Mathematics Dept., University of California, Berkeley, CA 94720; email: wkahan@cs.berkeley.edu; X. S. Li, Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720; email: xsli@lbl.gov; S. Mukherjee, Oracle, 100 Oracle Parkway, Redwood Shores, CA 94065; email: sonil.mukherjee@oracle.com; E. J. Riedy, Computer Science Division, University of California, Berkeley, CA 94720; email: ejr@cs.berkeley.edu.

We report extensive test results on over 6.2 million matrices of dimensions 5, 10, 100, and 1000. As long as a normwise (componentwise) condition number computed by the algorithm is less than $1/\max\{10,\sqrt{n}\}\varepsilon_w$, the computed normwise (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 $\varepsilon_w = 2^{-24}$ is the working precision. Residuals were computed in double precision (53 bits of precision). In other words, the algorithm always computed a tiny error at negligible extra cost for most linear systems. For worse conditioned problems (which we can detect using condition estimation), we obtained small correct error bounds in over 90% of cases.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Linear systems (direct and iterative methods), Error analysis*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Linear algebra, LAPACK, BLAS, floating-point arithmetic

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.

Algorithm 1: Basic iterative refinement

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 $x^{(i+1)} = x^{(i)} - dx^{(i+1)}$
  $i = i + 1$
untill $x^{(i)}$ is "accurate enough"
return $x^{(i)}$ and an error bound

(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 a rounding error in the inner loop prevents immediate convergence and makes the behavior and analysis interesting.

Mixed precision iterative refinement was first used by Wilkinson [1948] and Snyder [1955]. Analysis of Wilkinson [Bowdler et al. 1966] and Moler [1967] show that, if the residual is computed to about double the working precision, then the solution $x^{(i)}$ will converge to roughly working precision as long as the condition number of A is not too large (sufficiently less than $1/\varepsilon_w$, where ε_w is the working precision). Furthermore, the cost of this high accuracy is negligible: $O(n^2)$ on top of the $O(n^3)$ cost of straightforward Gaussian elimination.

With the exception of NAG Library [NAG Ltd 2005], this attractive algorithm has not been widely adopted. Its conspicuous omission from other standard libraries such as LAPACK [Anderson et al. 1999] or its predecessor LINPACK [Dongarra et al. 1979] can be attributed to two obstacles: (1) the lack of a portable implementation of higher precision arithmetic to compute the residual, and (2) the lack of a good way to compute a reliable error bound for the corrected solution. The recent completion of the BLAS Technical Forum Standard [Forum 2002a, 2002b] has essentially eliminated the first obstacle, and our goal in this article is to eliminate the second obstacle.

Section 2 summarizes the error analysis of Algorithm 1. This analysis justifies our stopping criterion and bound for the *normwise relative error*

$$B_{\text{norm}} \stackrel{\text{def}}{=} \frac{\|x^{(i)} - x\|_\infty}{\|x\|_\infty}. \tag{1}$$

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

Note that Algorithm 1 is *column scaling invariant*. More precisely, if we assume that (1) our basic solution scheme is GEPP without any Strassen-like implementation [Strassen 1969], (2) no overflow or 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 IEEE-754 floating-point standard arithmetic [ANSI/IEEE 1985]), then replacing the matrix A by $A_c \stackrel{\text{def}}{=} AC$ results in *exactly* the same roundoff errors being committed: 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 floating-point approximation satisfies $x_c^{(i)} = C^{-1}x^{(i)}$ exactly.

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 any scaled system $A_c x_c = b$. Thus at a modest extra cost, we can 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.) Using this, we can 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 is $C \approx \text{diag}(x)$ so that each component $x_{c,j} \approx 1$. Then the scaled relative error (2) becomes 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 at least partially (since C, which is computed on-the-fly, will affect the stopping criterion too).

Section 2 summarizes the precise stopping criterion and error bound for Algorithm 1. This analysis leads to two condition numbers that predict the success of iterative refinement. Let n be the matrix dimension and ε_w be the working precision. Error analysis shows that, if the normwise condition number $\kappa_{\text{norm}}(A)$ does not exceed $1/\gamma\varepsilon_w$ (where γ is a function of n), then the algorithm converges

with small normwise error and error bound. Similarly, if the componentwise condition number $\kappa_{\mathrm{comp}}(A)$ does not exceed $1/\gamma\varepsilon_w$, then we obtain convergence with small componentwise error and error bound. The following numerical experiment described below confirms this result and gives an empirical value $\gamma = \max\{10, \sqrt{n}\}$.

Section 3 describes our ultimate algorithm, Algorithm 2. This algorithm differs from Algorithm 1 in several important ways besides computing both normwise and componentwise error bounds. In particular, if consecutive corrections $dx^{(i)}$ are not decreasing rapidly enough, the algorithm increases the precision of the solution by switching to represent $x^{(i)}$ in *doubled working precision*, that is, by a pair of working precision arrays representing (roughly) the leading and trailing bits of $x^{(i)}$ as though it were in double precision. Iteration continues subject to the same progress monitoring scheme. This significantly improves componentwise accuracy on the most ill-conditioned problems.

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.

We summarize the results of these numerical tests. All of our experiments are conducted using IEEE-754 single precision as the working precision ($\varepsilon_w = 2^{-24}$). IEEE-754 double precision is used for residual computation ($\varepsilon_r = 2^{-53}$). First we consider the normwise error and error bound. For problems that are not too ill-conditioned (those with $\kappa_{\mathrm{norm}}(A) < 1/\gamma\varepsilon_w$), Algorithm 2 always computed an error bound of at most $2\gamma\varepsilon_w$, which exceeded the true error. For even more ill-conditioned problems (with κ_{norm} ranging from $1/\gamma\varepsilon_w$ to ε_w^{-2}), Algorithm 2 still gets similarly small normwise error bounds and true errors in 96% of cases. Note that we can detect this extreme ill-conditioning using condition estimators.

Next we consider the componentwise error and error bound. For problems that are not too ill-conditioned (those with $\kappa_{\mathrm{comp}}(A) < 1/\gamma\varepsilon_w$), Algorithm 2 always computed an error bound of at most $2\gamma\varepsilon_w$, which again exceeded the true error. For these problems the number of iterations required was at most 4, with a median of 2. For even more ill-conditioned problems, with componentwise condition numbers κ_{comp} ranging up past ε_w^{-2}, Algorithm 2 still gets similarly small componentwise error bounds and true errors in 94% of cases. As in the normwise case, this extreme ill-conditioning is detected using condition estimators.

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 [Forum 2002a, 2002b] for which portable implementations exist [Li et al. 2002]. 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 as in Li et al. [2002]. Even without this second operation, our algorithm manages to obtain small true errors and error bounds for problems that are not too ill-conditioned.

The rest of this article is organized as follows. Section 2 summarizes the error analysis of our algorithm, including their invariance under column scaling (see Demmel et al. [2004, Section 2] for details). Section 3 describes the ultimate algorithm, Algorithm 2. Section 4 describes related work. Section 5 describes the testing configuration, including how test matrices are generated. Section 6 presents the results of numerical tests. Finally, Section 7 draws conclusions and describes future work.

2. ERROR ANALYSIS

In this section, we give a summary of the error analysis used to justify our choices for termination criteria, error estimates, and the accuracy with which each step is performed. For details see Demmel et al. [2004, Section 2].

Let the true forward error of iteration i be $e^{(i)} \stackrel{\text{def}}{=} x^{(i)} - x$. Our main contributions are (1) to show that the termination criteria and error estimates apply not just to $\|e^{(i)}\|_\infty$ but to any diagonally scaled error $\|Ce^{(i)}\|_\infty$, and (2) to derive an estimated condition number (depending on C) that can be used to identify linear systems when convergence to an accurate solution is guaranteed as opposed to extremely ill-conditioned cases where such convergence is only likely.

Our algorithm uses three different precisions.

—ε_w is the working precision used to store the input data A and b. The factorization of A is also carried out in this precision. In our numerical experiments, we use IEEE-754 single precision as the working precision ($\varepsilon_w = 2^{-24}$).

—ε_x is the precision used to store the computed solution $x^{(i)}$. It is at least the working precision, possibly twice the working precision ($\varepsilon_x \le \varepsilon_w^2$) if necessary for componentwise convergence. In our numerical experiments, we use IEEE-754 single initially and switch to doubled single precision ($\varepsilon_x = \varepsilon_w^2 = 2^{-48}$) if required for componentwise convergence. The criteria for choosing ε_x is discussed later.

—ε_r is the precision used to compute the residuals $r^{(i)}$. We usually have $\varepsilon_r \ll \varepsilon_w$, typically at least twice the working precision ($\varepsilon_r \le \varepsilon_w^2$). In our numerical experiments, we use IEEE-754 double precision as the working precision ($\varepsilon_w = 2^{-53}$).

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

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

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

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

Absolute values of matrices and vectors are interpreted elementwise.

Classical analysis in Bowdler et al. [1966] and Moler [1967] show that, if we ignore the errors $\delta r^{(i)}$ and $\delta x^{(i+1)}$, then the correction $\|dx^{(i+1)}\|_\infty$ decreases by a factor of at most $\rho = O(\varepsilon_w)\|A\|_\infty\|A^{-1}\|_\infty$ at every step. Thus if $\kappa_\infty(A) \stackrel{\text{def}}{=} \|A\|_\infty\|A^{-1}\|_\infty$ is sufficiently less than $1/\varepsilon_w$ (so that $\rho < 1$), then the solution

$x = x^{(1)} - \sum_{j=2}^{\infty} dx^{(j)}$ converges like a geometric sum, so that

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

We can conservatively estimate the rate of convergence by taking the maximum ratio of successive correction: $\rho_{\max} \stackrel{\text{def}}{=} \max_{j \le i} \frac{\|dx^{(j+1)}\|_\infty}{\|dx^{(j)}\|_\infty}$. This estimated rate leads to a crude bound for the norm of the error

$$\|e^{(i)}\|_\infty \le \frac{\|dx^{(i+1)}\|_\infty}{(1-\rho_{\max})} .$$

This crude bound is the basis of our computed error bound. The bound will be reliable as long as convergence is fast enough, that is, $\rho_{\max}$ does not exceed some $\rho_{\text{thresh}} < 1$. Wilkinson [1963] chose $\rho_{\text{thresh}} = \frac{1}{2}$ which we use for our cautious mode. Our normwise error bound is thus

$$B_{\text{norm}} \stackrel{\text{def}}{=} \max \left\{ \frac{\|dx^{(i+1)}\|_\infty / \|x^{(i)}\|_\infty}{1-\rho_{\max}}, \gamma \varepsilon_w \right\} \approx \frac{\|x^{(i)} - x\|_\infty}{\|x\|_\infty} \stackrel{\text{def}}{=} E_{\text{norm}}, \tag{6}$$

where $\gamma = \max(10, \sqrt{n})$ has been chosen based on our numerical experiments to account for dependence of error on dimension.

Geometric convergence will cease when the rounding errors $\delta r^{(i)}$ and $\delta x^{(i+1)}$ become significant. We detect this by checking whether the ratio $\|dx^{(i+1)}\|_\infty / \|dx^{(i)}\|_\infty$ of successive corrections exceeds the threshold ρ_{thresh}.

The refinement will stop if *any* of the following three condition applies:

(1) $\frac{\|dx^{(i+1)}\|_\infty}{\|x^{(i)}\|_\infty} \le \varepsilon_w$ (the correction $dx^{(i+1)}$ changes solution $x^{(i)}$ too little)

(2) $\frac{\|dx^{(i+1)}\|_\infty}{\|dx^{(i)}\|_\infty} \ge \rho_{\text{thresh}}$ (convergence slows down sufficiently)

(3) $i > i_{\text{thresh}}$ (too many iterations have been performed)

When the refinement stops, we return an error bound based on (6).

We now discuss diagonal scaling of A. Existing linear equation solvers may already *equilibrate*, that is, replace the input matrix A by $A_s = R \cdot A \cdot C$, where R and C are diagonal matrices chosen to try to make $\kappa_\infty(A_s)$ much smaller than $\kappa_\infty(A)$. This scaling changes the linear system from $Ax = b$ to $A_s x_s = b_s$, where $x_s = C^{-1}x$ and $b_s = Rb$. We perform iterative refinement on the scaled matrix A_s.

If we choose diagonal entries of R and C to be powers of the floating-point radix β, then no additional rounding error is introduced by factoring A_s rather than A. R has no effect on x or the way we measure error but can affect the pivot choices during factorization as well as the convergence rate and the scaled condition number. In contrast, C affects only the norm in which we measure the error of x.

Iterative refinement on the system $A_s x_s = r_s$ produces a sequence of scaled corrections $dx_s^{(i)}$ and solutions $x_s^{(i)}$. But conventional implementations of LU decomposition are column-scaling independent, relating these sequences to corrections and solutions of the original system by

$$dx_s^{(i)} = C^{-1} dx^{(i)} \qquad \text{and} \qquad x_s^{(i)} = C^{-1} x^{(i)} .$$

These relations are exact (ignoring over and underflow) when R and C are restricted to powers of the radix and the factorization is column-scaling invariant. Thus we can evaluate our error bound and stopping criteria with the scaled quantities $Cdx_s^{(i)}$ and $Cx_s^{(i)}$ to determine these quantities for the unscaled system. Computation of each scaled norm requires $O(n)$ additional work which is insignificant compared to $O(n^2)$ work required to compute the residual at each step. The corresponding condition number governing convergence is

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

which is easy to estimate using standard condition estimation techniques [Higham 1987, 1988, 1990]. By choosing R to nearly equilibrate the rows of A, we can minimize κ_{norm} to roughly $\text{cond}(A) \stackrel{\text{def}}{=} \left\| |A^{-1}||A| \right\|_\infty$, the Skeel condition number.

We restrict the equilibration scalings R and C to powers of the radix to prevent additional errors during factorization. However, in interpreting the error bounds and termination criteria, we are free to chose any additional scaling matrix $\hat{C}$. Rounding errors that occur when computing $\|\hat{C}x_s^{(i)}\|_\infty$ and $\|\hat{C}dx_s^{(i)}\|_\infty$ contribute at most a relative error of ε_w to each quantity and hence are negligible. If we could choose $\hat{C} = \text{diag}(x_s)$, then combining the two scalings gives $C\hat{C} = \text{diag}(x)$. Then we would have the normwise relative error $\|\hat{C}^{-1}x_s^{(i)} - \hat{C}^{-1}x_s\|_\infty = \|\hat{C}^{-1}C^{-1}x^{(i)} - \hat{C}^{-1}C^{-1}x\|_\infty = \|(C\hat{C})^{-1}x^{(i)} - (C\hat{C})^{-1}x\|_\infty = \max_k |x_k^{(i)} - x_k||x_k|$. So the normwise error in $\hat{C}^{-1}x_s^{(i)}$ is equal to the componentwise relative error of the original system. We exploit this relation to evaluate componentwise error bounds, stopping criteria, and condition number.

Since computing $\hat{C}$ requires the solution x_s, we can at best approximate it. An initial solution can be far from the truth, so our final algorithm monitors the convergence of each component of $x_s^{(i)}$. Once each component has settled down to at least 2 bits (i.e., a relative error of $1/4$), we use the computed $x^{(i)}$ to approximate $\hat{C} = \text{diag}(x_s)$. This leads to the componentwise error bound

$$B_{\text{comp}} \stackrel{\text{def}}{=} \max\left\{ \frac{\|\hat{C}dx_s^{(i)}\|_\infty}{1-\hat{\rho}_{\max}}, \gamma\varepsilon_w \right\} \approx \max_k \left| \frac{x_k^{(i)} - x_k}{x_k} \right| \stackrel{\text{def}}{=} E_{\text{comp}}, \tag{7}$$

where $\hat{\rho}_{\max} = \max_{j \le i} \frac{\|\hat{C}dx_s^{(j+1)}\|_\infty}{\|\hat{C}dx_s^{(j)}\|_\infty}$ is the estimate of the convergence rate of $\hat{C}^{-1}x_s^{(i)}$. The componentwise condition number governing this convergence is given by

$$\kappa_{\text{comp}} = \kappa_\infty(A_s \cdot \hat{C}) = \kappa_\infty(R \cdot A \cdot \text{diag}(x)).$$

We note that the same technique may be applied to LAPACK's current working precision refinement algorithm in xGERFS. Scaling by $\hat{C} \approx \text{diag}(x)$ in LAPACK's forward error estimator produces the loose componentwise error estimate

$$B_{\text{comp}} = \|\hat{C}^{-1} \cdot |A^{-1}| \cdot \left(|r| + (n+1)\varepsilon_w(|A||x_s| + |b|) \right) \|_\infty. \tag{8}$$

3. ALGORITHMIC DETAILS

We now describe our ultimate iterative refinement procedure, Algorithm 2. Its cost amounts to a small number of triangular solves and matrix-vector

Algorithm 2: New iterative refinement

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 $\kappa_s = \kappa_\infty(A_s)$
Solve $A_s y^{(1)} = b_s$ using the basic solution method
5 $\|dx^{(1)}\| = \|dz^{(1)}\| =$ final-relnorm$_x =$ final-relnorm$_z = \infty$
$\rho_{\max,x} = \rho_{\max,z} = 0.0$, x-state = working, z-state = unstable, y-scheme = single
7 for $i = 1$ to i_{thresh} do
// **Compute residual in precision** ε_r
if y-scheme = single $r^{(i)} = A_s y^{(i)} - b_s$
10 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 $\|dz^{(i+1)}\| = \max_j |dy_j^{(i+1)}/y_j^{(i)}|$
15 if y-scheme = single and $\kappa_s \cdot \max_j |y_j| / \min_j |y_j| \geq 1/\gamma\varepsilon_w$ **then** incr-prec = true
16 Update x-state, $\rho_{\max,x}$ with Procedure `new-x-state` below
17 Update z-state, $\rho_{\max,z}$ with Procedure `new-z-state` below
// **Either update may signal** incr-prec **or may set its** final-relnorm
19 if x-state $\neq$ working and z-state $\neq$ working **then BREAK**
20 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
24 if x-state = working then final-relnorm$_x = \|dx^{(i+1)}\|/\|x^{(i)}\|$
25 if z-state = working then final-relnorm$_z = \|dz^{(i+1)}\|$
return $x^{(i)} = Cy^{(i)}$,
normwise error bound $\max\{\frac{1}{1-\rho_{\max,x}} \cdot$ final-relnorm$_x$, $\max\{10, \sqrt{n}\} \cdot \varepsilon_w\}$, and
componentwise error bound $\max\{\frac{1}{1-\rho_{\max,z}} \cdot$ final-relnorm$_z$, $\max\{10, \sqrt{n}\} \cdot \varepsilon_w\}$

multiplications on top of Gaussian elimination. For a dense matrix, the cost is $O(n^2)$ on top of the $O(n^3)$ required for the solution without refinement.

The algorithm refers to auxiliary vectors $y \stackrel{\text{def}}{=} x_s$, $dy \stackrel{\text{def}}{=} dx_s$, $z \stackrel{\text{def}}{=} \hat{C}x_s$ and $dz \stackrel{\text{def}}{=} \hat{C}dx_s$. The actual implementation only stores y and dy to minimize storage; x, dx, z, and dz are not stored explicitly. All norms in this section are the infinity norm. The state variable x-state $\in$ {working, no-progress, converged} tracks the normwise convergence (convergence of x), while z-state $\in$ {unstable, working, no-progress, converged} tracks the componentwise convergence (convergence of z to vector of all 1's). State variable y-scheme $\in$ {singledouble} denotes the precision used for storing y (starting out with single, switching to double if necessary).

The refinement loop exits either from a large iteration count or when both x and z are no longer in a working state (line 19). The logic is somewhat complicated by three facts. First, we are simultaneously applying two sets of stopping criteria, one for the normwise error and one for componentwise error. Thus the algorithm must decide what to do when one set of criteria decides to

stop (either because of convergence or failure to converge) before the other set of criteria; basically we continue if we are making progress in either normwise or componentwise sense.

Second, the algorithm must decide when to switch to storing the solution y in doubled working precision. To do this, a second vector y_t is used to store the trailing bits of each floating-point component so that $y + y_t$ (with $|y_t| \ll |y|$) represents the current solution. Storing y in doubled working precision is more costly so it is only used when progress stops before reaching full convergence or when a cheap upper bound on κ_{comp} is very large (line 15).

Third, we must decide when each component of the solution has settled down enough to test for componentwise relative convergence. We begin considering componentwise error if no component's relative change exceeds a threshold dz_{thresh}, empirically set to $1/4$.

We must also take precautions to return reasonable error bounds when some solution components are exactly zero because of the sparsity structure of A and b. For example, if $b = 0$, then the solution $x = 0$ should be returned with a zero error bound. See Demmel et al. [2004, Section 7.3] for further discussion.

```
Input: Current x-state, ||x^(i)||, ||dx^(i)||, ||dx^(i-1)||, y-scheme
Output: New x-state and ρ_max,x, possibly signaling incr-prec or updating
        final-relnorm_x
if x-state = no-progress and ||dx^(i+1)||/||dx^(i)|| ≤ ρ_thresh then x-state = working
if x-state = working then
    if ||dx^(i+1)||/||x^(i)|| ≤ ε_w then x-state = converged else if ||dx^(i+1)||/||dx^(i)|| >
    ρ_thresh then
        if y-scheme = single then incr-prec = true
        else x-state = no-progress
    else ρ_max,x = max{ρ_max,x, ||dx^(i+1)||/||dx^(i)||}
    if x-state ≠ working then final-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 ρ_max,z, possibly signaling incr-prec or updating
        final-relnorm_z
if z-state = unstable and ||dz^(i+1)|| ≤ dz_thresh then z-state = working
if z-state = no-progress and ||dz^(i+1)||/||dz^(i)|| ≤ ρ_thresh then z-state = working
if z-state = working then
    if ||dz^(i+1)|| ≤ ε_w then z-state = converged
    else if ||dz^(i+1)|| > dz_thresh then
        z-state = unstable, final-relnorm_z = ∞, ρ_max,z = 0.0
    else if ||dz^(i+1)||/||dz^(i)|| > ρ_thresh then
        if y-scheme = single then incr-prec = true
        else z-state = no-progress
    else ρ_max,z = max{ρ_max,z, ||dz^(i+1)||/||dz^(i)||}
    if z-state ≠ working then final-relnorm_z = ||dz^(i+1)||
```

Procedure new-z-state

4. RELATED WORK

Mixed precision iterative refinement was first used by Wilkinson [1948] and Snyder [1955] although Mallock's work [1933] seems to imply an analog electrical circuit implementing working precision iterative refinement. In Bowdler et al. [1966], Wilkinson et al. present the Algol programs that perform the LU factorization, the triangular solutions, and the iterative refinement using $\varepsilon_r = \varepsilon_w^2$. Wilkinson's LU factorization uses a Crout algorithm, where the inner products are computed to extra precision but in our numerical experiment we use the same implementation of Gaussian elimination from LAPACK as Algorithm 2.

Wilkinson's algorithm differs from ours in several ways. (1) There is no initial equilibration. (2) Parameter ρ_{thresh} is fixed to 0.5. (3) The solution vector x is stored only to working precision. (4) Wilkinson's algorithm does not attempt to achieve componentwise accuracy. (5) The original paper's algorithm [Bowdler et al. 1966] does not return an error bound, though an error analysis appears in Wilkinson [1963] and Moler [1967]. In particular, Moler [1967, Equation (11)] and Higham [2002, Theorem 12.1] provide error bounds including several functions of problem dimension, condition number, and other quantities that are hard to estimate tightly and efficiently. For the sake of later numerical comparisons, we supplement Wilkinson's algorithm with the error bounds and stopping criteria described in Section 2. We refer to the combination as Algorithm W.

The use of higher precision in storing x was first presented as an exercise in Stewart's book [1973, 206–207]. Stewart suggests that, if x is accumulated in higher and higher precision, then the residual will get progressively smaller, and the iteration will eventually converge to any desired accuracy. Kielbasiński [1981] proposes an algorithm called binary cascade iterative refinement. In this algorithm, GEPP and the first triangular solve for $x^{(0)}$ are performed in a base precision. Then during iterative refinement, $r^{(i)}$, $x^{(i+1)}$ and $dx^{(i)}$ are computed in successively higher precision. Kielbasiński shows that, with a prescribed accuracy for x, you can choose a maximum precision required to stop the iteration. This requires arbitrary precision arithmetic. We are not aware of any implementation of this algorithm.

A different approach to guaranteeing accuracy of a solution is to use interval arithmetic [Rump 1983, 1995]. We will not consider interval algorithms further since the available algorithms do not meet our criterion of costing a small extra delay on top of the fastest available solution method.

Björck [1990] gives a nice survey of iterative refinement for linear systems and least-squares problems, including error estimates using working precision or extra precision in residual computation. Higham's book [2002] 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 $\left\| |A^{-1}| \cdot |A| \cdot |x| \right\|_\infty$ and require estimating the norm of A^{-1}. We experimented with approximating these error estimates without using refinement within the norm estimator (which requires the solution of linear systems with A and A^T), but they did not provide more accurate error bounds.

Extra precise iterative refinement has not been adopted in such standard libraries as LINPACK [Dongarra et al. 1979] and later LAPACK [Anderson et al. 1999] mainly because there was no portable way to implement extra precision when the working precision was already the highest precision supported by the compiler. A notable exception is the NAG Library [NAG Ltd 2005] whose F04ATF routine implements extra-precise iterative refinement (but does not provide any error bound). 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 [Higham 2002, Theorem 12.3], the LAPACK refinement routines use the componentwise backward error in the stopping criteria. For the sake of later numerical comparisons, we augment this algorithm as described in Section 2 to compute a componentwise error bound. This combination is Algorithm L.

5. TESTING CONFIGURATION

5.1 Hardware and Software Platforms

The XBLAS library [Li et al. 2002] 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 [Forum 2002a, 2002b]). 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 $y \leftarrow \alpha Ax + \beta y$ in single, double, indigenous, or extra precision.

In addition to the extra precision routines provided by the XBLAS, the doubled-x scheme in Algorithm 2 requires a new routine which we call `gemv2`. This routine takes a matrix A, three vectors x, y, and z, and two scalars α and β to compute $z \leftarrow \alpha A(x + y) + \beta z$ in single, double, indigenous, or extra precision. This routine enables us to compute an accurate residual (computed with precision ε_r) when the solution is kept in two words x and x_t: $r = A(x + x_t) - b$ (see line 10 in Algorithm 2).

This new routine is implemented and tested following the same strategy used in the XBLAS reference implementation [Li et al. 2002]. Note that this routine is only used when Algorithm 2 switches to storing the solution x in doubled working precision. Even without this new routine, our algorithm manages to obtain small errors and error bounds for problems that are not too ill-conditioned. See Demmel et al. [2004, Section 6.4.1] for how the precision of x affects results.

We tested our code on two platforms: Sun UltraSPARC 2i running Solaris and Itanium 2 running Linux. The results were largely similar so we present results only from the Itanium 2. For more details regarding compilers and BLAS libraries used, see Demmel et al. [2004, Section 5.5].

5.2 Test Matrix Generation

To thoroughly test our algorithms, we need many test cases with a wide range of condition numbers, various distribution of singular values, well-scaled and

ill-scaled matrices, matrices with first k columns nearly linearly dependent (so that ill-conditioning causes the k-th pivot to be tiny), and a wide range of solution component sizes. 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. The range chosen will generate both easy and hard problems (since $\varepsilon_w = 2^{-24}$) independent of scaling.

(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 randomly from $\{3, n/2, n\}$. Move the largest and the smallest singular values (picked in step 1) into the first k values, and let Σ be the resulting diagonal matrix. let

$$\tilde{A} = U \Sigma \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix}, \tag{9}$$

   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 leading k columns of $\tilde{A}$ nearly linearly dependent, so that LU factorization will most likely encounter a small pivot at the k-th step. U, V_1 and V_2 are applied via sequences of random reflections of dimensions 2 through n, k or $n-k$, respectively.

(4) Pick τ with $(\log_2 \tau)^{1/2}$ uniformly distributed in $[0, \sqrt{24}]$.[1] We generate $\tilde{x}$ so that $\tau = \frac{\max_i |\tilde{x}_i|}{\min_i |\tilde{x}_i|}$. 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) $\tilde{x_i}$'s are randomly chosen within $[\tau^{-1}, 1]$ so that $\log \tilde{x_i}$ is uniformly distributed.

   If one of the first four distributions is chosen, we multiply $\tilde{x}$ by a uniform random number in $[0.5, 1.5]$ to make the largest element unequal to 1 (so that all components of $\tilde{x}$ have significands with many bits equal to 1).

(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 $[0, \sqrt{24}]$.[2] 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 but rounded to single at the end (using the XBLAS routine `BLAS_sgemm_x`).

[1] 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 $\kappa_{\text{comp}} = \kappa(RA\,\text{diag}(x)) = \kappa(A_s\,\text{diag}(y))$.

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

(7) Compute $x = A^{-1}b$ by using double precision GEPP with double-double precision iterative refinement. This corresponds to Algorithm 2 with IEEE-754 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 the original $\bar{x}$ because of rounding errors committed in step (6). This difference can be quite large if A is ill-conditioned.

Our tests include linear systems generated as just presented of dimensions 5, 10, 100, and 1000. We use 2×10^6 systems for each dimension except 1000. We test only 2×10^5 systems of dimension 1000 both to save time and because larger random systems are less likely to show aberrant behavior. Statistics for the systems of dimension 100 are presented next.

5.3 Test Matrix Statistics

To make sure that we had a good (if not entirely uniform) sampling of easy and hard problems, various condition numbers were defined and plotted. See Demmel et al. [2004, Figures 1 and 2] for these plots. We present these condition numbers in the following.

(1) $\kappa_s = \kappa_\infty(RAC) = \kappa_\infty(A_s)$ is 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 57 to 1.6×10^{13}, with all but 2169 (0.11%) smaller than 10^{10}.

(2) $\kappa_c = \kappa_\infty(C) = \|C\|_\infty$ is the maximum column-scaling factor computed during equilibration. This varies from 1 to $2^{35} \approx 3.4 \times 10^{10}$.

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

(4) $\kappa_y = \kappa_\infty(\mathrm{diag}(y)) = {}^{\max_i |y_i|}/_{\min_i |y_i|}$ is the ratio between the largest and smallest element (in magnitude) of $y = C^{-1}x$, the scaled solution of the equilibrated system $A_s y = b_s$. κ_y 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 vary, 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 $\kappa_{\mathrm{comp}} = \kappa_\infty(A_s\,\mathrm{diag}(y)) \le \kappa_s\kappa_y$.

(5) $\kappa_{\mathrm{norm}} = \kappa_\infty(A_s \cdot C^{-1}) = \kappa_\infty(R \cdot A)$ is the normwise condition number. It varies from about 57 to 7.6×10^{17}. Using κ_{norm} we divide the test matrices into two categories:

—*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 after refinement in the normwise sense. Most problems can be expected to fall in this category in practice. Of the two million test matrices, 821,097 cases fall into this category.

—*Normwise ill-conditioned problems*. These are the matrices with $\kappa_{\mathrm{norm}} > {}^{1}/_{\gamma\varepsilon_w}$. These are so ill-conditioned that we cannot guarantee accurate solutions.

Of the two million test matrices, 1,178,903 cases are in 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. $\gamma = \max\{10, \sqrt{n}\}$ both protects against condition number underestimates and keeps the bounds attainable.

(6) $\kappa_{\text{comp}} = \kappa(RA\,\text{diag}(x)) = \kappa(A_s\,\text{diag}(y))$ is the condition number for componentwise problem. It varies from about 57 to 4.5×10^{15}. 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 before, we use κ_{comp} to divide the problems into two categories:

—*Componentwise well-conditioned problems*. These are problems with $\kappa_{\text{comp}} \leq 1/\gamma\varepsilon_w$, 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, 545,427 cases fall into this category.

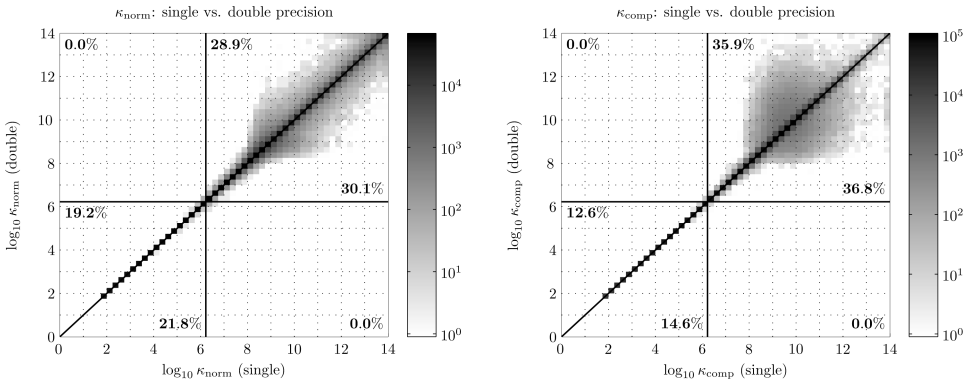
—*Componentwise ill-conditioned problems*. These are the matrices with $\kappa_{\text{comp}} > 1/\gamma\varepsilon_w$. These are so ill-conditioned that we cannot guarantee accurate solutions. Of the two million test problems, 1,454,573 cases fall into this category. Note that if any component of the solution is zero, then κ_{comp} becomes infinite.

For the basic solution method, we used LAPACK's `sgetrf` (GEPP, $PA_s = LU$) and `sgetrs` (triangular solve) routines. The pivot growth factor $\frac{\max_{i,j}|U(i,j)|}{\max_{i,j}|A_s(i,j)|}$ never exceeded 72. This implies that we have obtained LU factors with small normwise backward error. The distribution of the pivot growth factors [Demmel et al. 2004, Figure 3] shows a smooth decrease in likelihood with increasing pivot growth. This is consistent with Trefethen and Schreiber [1990] which suggests that, for various random matrix distributions, the average pivot growth factor should be about $n^{2/3} \approx 22$.

5.4 Accuracy of Single Precision Condition Numbers

All the condition numbers in Section 5.3 were estimated in double precision which we regard as truth. Since Algorithm 2 will only estimate the condition number in working (single) precision, we must confirm whether the single precision condition number is a reliable indicator for detecting ill-conditioned matrices.

The 2D histograms of the normwise condition number κ_{norm} computed in single precision and double precision, are shown in Figure 1(a). Since these 2D histograms appear throughout this article, we explain this plot in some detail. In this 2D histogram, each shaded square at coordinate (x, y) indicates the existence of matrices that have single precision κ_{norm} in the range $[10^x, 10^{x+1/4})$ and double precision κ_{norm} in the range $[10^y, 10^{y+1/4})$. The shade of each square indicates the number of matrices that fall in that square, indicated by the



(a) κ_{norm} (b) κ_{comp}

Fig. 1. Accuracy of computed condition numbers: single precision vs. double precision. The single precision κ_{comp} are those computed by Algorithm 2 with $\rho_{\text{thresh}} = 0.5$.

grayscale bar to the right of the plot. Dark squares indicate a large population, while light squares indicate a very small population. A logarithmic scale is used in the grayscale bar so a lighter shade indicates a much smaller population than a darker shade. For example, the light squares near the top and right edges contains only a handful of matrices (less than 5, usually just 1), while the darker squares contains approximately 10^3 to 10^4 samples. These histograms can be interpreted as a test matrix population density at each coordinate.

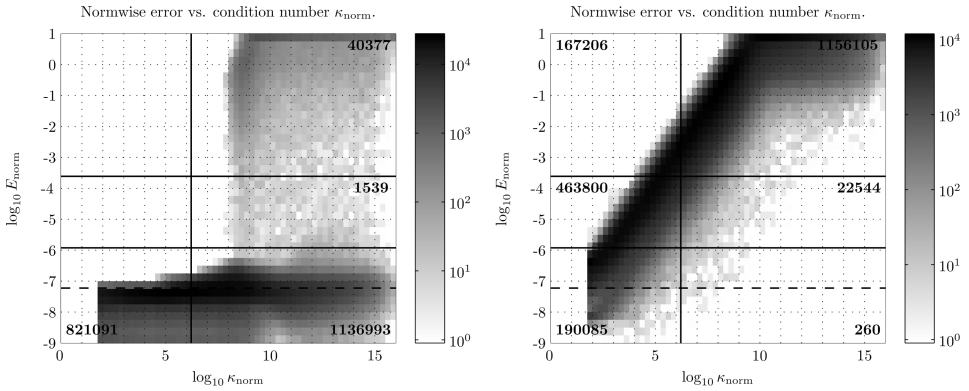
The horizontal and vertical lines are both located at $1/\gamma\varepsilon_w$, separating well-conditioned from ill-conditioned matrices. Along with the diagonal line where both single and double precision values of κ_{norm} are equal, these lines divide the plot into six regions, and the percentage of samples in each region is displayed in bold numbers in the plot.

If the single and double precision values of κ_{norm} were identical, the only shaded squares would be along the diagonal. When $\kappa_{\text{norm}} \leq 1/\gamma\varepsilon_w$, the squares are all close to the diagonal, meaning that the single precision κ_{norm} is close to the double precision κ_{norm} (which we assume is the true value). Only when either condition number exceeds $1/\gamma\varepsilon_w$ can they differ significantly as indicated by the spread of shaded squares in the upper right. This tells us that we can trust the single precision κ_{norm} to separate the not-too-ill-conditioned matrices from the very ill-conditioned matrices.

Figure 1(b) tells the same story for the componentwise condition number κ_{comp}.

6. NUMERICAL RESULTS

We present the numerical results for our new algorithm, Algorithm 2, and compare it to Algorithm W (Wilkinson's algorithm augmented with our error bound formulas) and Algorithm L (the current LAPACK algorithm augmented with our componentwise error bound (8). The data is for two million 100×100 test matrices described in Section 5. Similar results were obtained on two



(a) Algorithm 2 (b) Algorithm L

Fig. 2. Normwise error vs. κ_{norm}.

million each of 5×5 and 10×10 matrices and also two hundred thousand 1000×1000 matrices.[3]

The normwise and componentwise true errors are denoted by $E_{\text{norm}} = {}^{\|x-\hat{x}\|_\infty}/_{\|x\|_\infty}$ and $E_{\text{comp}} = \max_i {}^{|x_i-\hat{x}_i|}/_{|x_i|}$, respectively, 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. Sections 6.1 and 6.2 present normwise and componentwise results, respectively. We set $\rho_{\text{thresh}} = 0.5$ in Algorithms 2, and W. Parameter i_{thresh} (maximum number of iterations allowed) is set very large so that we can evaluate the number of steps required to converge.

Here we define notation common to Sections 6.1 and 6.2. For example, for E_{norm} we distinguish three cases.

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

(2) *Weak Convergence*. $2\gamma\varepsilon_w < E_{\text{norm}} \leq \sqrt{\varepsilon_w}$. We could not get strong convergence, but we did get at least half of the digits correctly. The upper solid horizontal line in Figure 2 (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.

In addition, we often indicate the value of ε_w in figures as well: the dashed horizontal line in Figure 2 (and in other analogous figures) is at $E_{\text{norm}} = \varepsilon_w$. Analogous classifications can also be made for B_{norm}, E_{comp}, and B_{comp}, and separating vertical or horizontal lines appear in various figures.

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. But in this section, we report the computed error bounds without setting them to 1, in order to better understand their behavior.

[3]A full set of color plots for all algorithms and all 6.2 M test matrices can be seen at `http://www.cs.berkeley.edu/~demmel/itref-data/`.

Later in Section 6.4, we will vary the parameters ρ_{thresh} and i_{thresh} to study how the behavior of Algorithm 2 changes. In particular, we will recommend cautious and aggressive values for i_{thresh} and ρ_{thresh}. The cautious settings, which we recommend as the default, yield maximally reliable error bounds for well-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

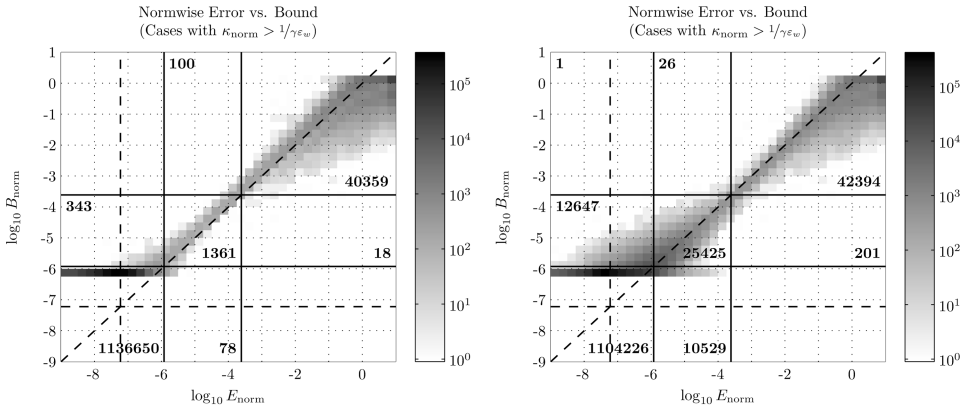
We start by looking at the results in terms of normwise true error E_{norm} and error bound B_{norm}. The most important conclusion we can draw from this section is that both Algorithm 2 and Algorithm W deliver a tiny error (E_{norm} strongly converged) and a slightly larger error bound (B_{norm} also strongly converged) as long as $\kappa_{\text{norm}} \leq {}^{1}/_{\gamma\varepsilon_w}$, that is, for all well-conditioned matrices in our test set. This is the best possible behavior we could expect. Furthermore, even for more ill-conditioned problems ($\kappa_{\text{norm}} > {}^{1}/_{\gamma\varepsilon_w}$), both algorithms still do very well; Algorithm 2 and W achieve strong convergence in both E_{norm} and B_{norm} in 96% and 92% of the cases, respectively.

The two plots in Figure 2 show the 2D histograms of the test problems plotted according to their true normwise error E_{norm} and condition number κ_{norm} for Algorithms 2 and L. The plot for Algorithm W is nearly identical to the plot for Algorithm 2 and so is omitted (see Demmel et al. [2004, Figure 6b]). The vertical solid line is at $\kappa_{\text{norm}} = {}^{1}/_{\gamma\varepsilon_w}$, and separates the well-conditioned problems from the ill-conditioned problems. The solid horizontal lines are at $\sqrt{\varepsilon_w}$ and $2\gamma\varepsilon_w$, separating the regions of strong and weak convergence of E_{norm}. The dotted horizontal line is at ε_w. If any problem falls outside the coordinate range, then it is placed at the edge; for example, the band at the very top of Figure 2 includes all the cases where $E_{\text{norm}} \geq 10$.

Figure 2(a) shows that for well-conditioned problems ($\kappa_{\text{norm}} < {}^{1}/_{\gamma\varepsilon_w}$), Algorithm 2 (and Algorithm W) attain the best possible result: strong convergence of E_{norm} in all cases (821,097 out of 2 million). Even for harder problems ($\kappa_{\text{norm}} \geq {}^{1}/_{\gamma\varepsilon_w}$), Algorithm 2 (and Algorithm W) still do very well, with Algorithm 2 exhibiting strong convergence of E_{norm} in 96% of cases and Algorithm W exhibiting strong convergence of E_{norm} in 93% of cases.

In contrast, with Algorithm L (Figure 2(b)), 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 [Higham 2002, Theorem 12.2]. Algorithm L consistently gives much larger true errors and error bounds than either of the other two algorithms even 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. For 821,097 well-conditioned problems, both Algorithms 2 and W converged to a true error of at most $3 \cdot 10^{-7}$ and returned an error bound of $\gamma\varepsilon_w \approx 6 \cdot 10^{-7}$, just slightly larger.



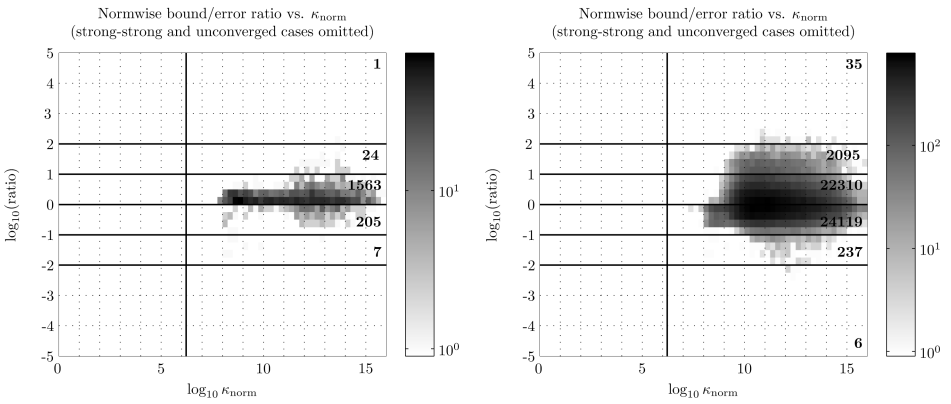
(a) Algorithm 2 (b) Algorithm W

Fig. 3. Normwise error vs. bound (ill-conditioned cases).

The 2D histograms in Figure 3 show what happens for the more ill-conditioned cases (the same plot for well-conditioned cases was omitted since all the cases would fall under the bottom left square delineated by the solid lines). The leftmost vertical solid lines are at $E_{\text{norm}} = 2\gamma\varepsilon_w$ (corresponding to the threshold for strong convergence) and the rightmost vertical solid lines are at $E_{\text{norm}} = \sqrt{\varepsilon_w}$ (corresponding to the threshold for weak convergence). Horizontal 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.

From this plot, we see that even for very ill-conditioned problems, algorithms yield strong convergence in both E_{norm} and B_{norm} in most cases. However our Algorithm 2 seems to give more cases near the diagonal (where $E_{\text{norm}} = B_{\text{norm}}$), leading to fewer overestimates and underestimates. Both algorithms fail to have B_{norm} converge in about the same number of cases, 3.4% and 3.6%, repectively. But of the remainder (those where both algorithms report at least weak convergence), Algorithm 2 fails to get strong convergence in both E_{norm} and B_{norm} in only 0.16% of the cases (1,800 out of 1,138,444 cases), whereas Algorithm W fails to get strong convergence in both E_{norm} and B_{norm} over 27 times as often, in 4.3% of the cases (4,8802 out of 1,136,482). In other words, there are over 27 times more cases where Algorithm W is confused than Algorithm 2 (48,802 versus 1,800).

Finally, Figure 4 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 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 2 and W converge strongly for both E_{norm} and B_{norm} for all



(a) Algorithm 2 (b) Algorithm W

Fig. 4. 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.

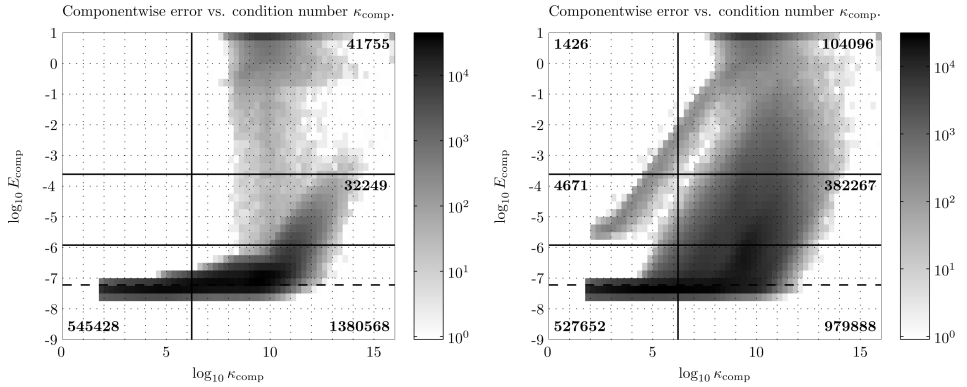
well-conditioned cases, no data points appear to the left of the vertical line. We are most concerned about underestimates where the error bound is substantially smaller than the true error. We see that Algorithm 2 has somewhat fewer underestimates than Algorithm W (defined as $E_{\text{norm}} > 10B_{\text{norm}}$), 7 vs. 243, and rather fewer overestimates ($10E_{\text{norm}} < B_{\text{norm}}$), 25 vs. 2,130. Analogous figures for Algorithm L (see Demmel et al. [2004, Figures 6c and 8c]) indicate that, while Algorithm L never underestimates the error, it almost always overestimates the error by two or three orders of magnitude. Thus the error bound returned by Algorithm L is loose, albeit safe.

6.2 Componentwise Error Estimate

We now look at the results in terms of componentwise true error E_{comp} and error bound B_{comp}. Algorithm W does not compute a componentwise error bound, nor was it designed to make E_{comp} small, so only its true error E_{comp} is analyzed in this section.

The most important conclusion we can draw from this section is that Algorithm 2 delivers a tiny componentwise error (E_{comp} strongly converged) and a slightly larger error bound (B_{comp} also strongly converged) as long as $\kappa_{\text{comp}} < 1/\gamma\varepsilon_w$, that is, for all componentwise well-conditioned matrices. This is the best possible behavior we could expect. Furthermore, even for harder problems where $\kappa_{\text{comp}} \geq 1/\gamma\varepsilon_w$, Algorithm 2 also does well, getting strong convergence in both E_{comp} and B_{comp} in 94% of the cases.

The two plots in Figure 5 show the 2D histograms of the test problems plotted according to their componentwise error E_{comp} and condition number κ_{comp} for the Algorithms 2 and W. These graphs may be interpreted similarly to those in Figure 2 which were described in the previous section. As already mentioned, Figure 5 shows that, for well-conditioned problems, Algorithm 2 attains strong convergence of E_{comp} in all cases. Algorithm W, which was not designed to get small componentwise errors, does slightly worse with strong convergence in



(a) Algorithm 2 (b) Algorithm W

Fig. 5. Componentwise error vs. κ_{comp}.

99% of the cases and weak or no convergence in the other 1%, including a few truly well-conditioned problems. For harder problems (those with $\kappa_{\text{comp}} \geq 1/\gamma\varepsilon_w$), Algorithm 2 still does very well, exhibiting strong convergence of E_{comp} in 95% of cases. Algorithm W does much worse, exhibiting strong convergence of E_{comp} in only 67% of cases and failing to converge at all more than twice as frequently. In contrast, with Algorithm L the error grows roughly proportional to the condition number, (see Demmel et al. [2004, Figure 9c]). Strong convergence of E_{comp} is very rare, only 7.3% of well-conditioned cases and not at all for ill-conditioned cases.

As in the last section, a small error E_{comp} is helpful only if the algorithm also produces a comparably small error bound B_{comp}. For well-conditioned problems, Algorithm 2 always obtains a small componentwise error bound around $\gamma\varepsilon_w$ and true error slightly smaller. Thus we can trust Algorithm 2 to deliver a tiny error and a slightly larger error bound as long as $\kappa_{\text{comp}} < 1/\gamma\varepsilon_w$.

The 2D histogram in Figure 6, whose interpretation is the same as that of Figure 3 in the previous section, shows what happens for the more ill-conditioned cases. Most (94%) still yield strong convergence in both E_{comp} and B_{comp}.

Finally, Figure 7 shows the 2D histogram of the ratio $B_{\text{comp}}/E_{\text{comp}}$ plotted against κ_{comp}. Again, this histogram may be interpreted similarly to those in Figure 2. However, in contrast to the normwise case, we see there are more cases where Algorithm 2 attains neither strong convergence (in both B_{comp} and E_{comp}) nor convergence failure in B_{comp}: 45,100 cases versus 1,800 (both out of 2 million, so rather few either way). There are also more cases of underestimates where the ratio is less than $1/10$, 273 vs. 7.

6.3 Iteration Counts

Table I shows the statistics on the number of iterations required by the algorithms. The parameter i_{thresh} has been set very large so that we can evaluate the number of steps required to converge. The data is broken down into well-conditioned problems versus ill-conditioned problems and by the number

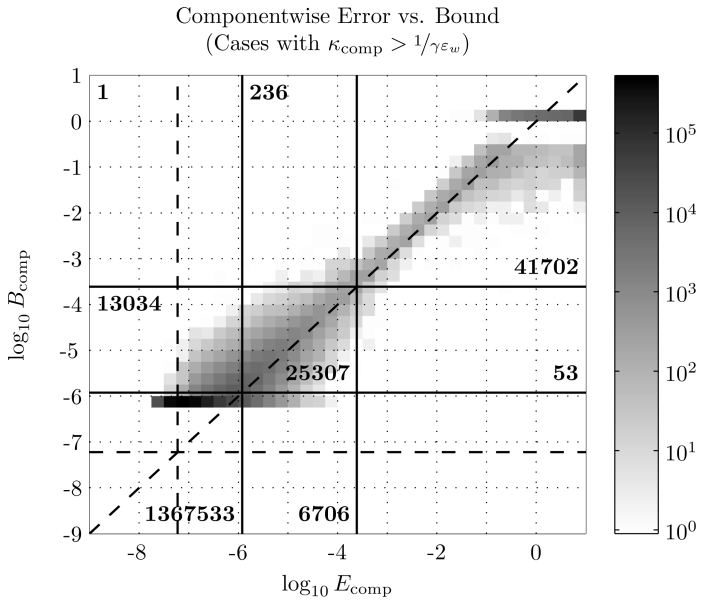


Fig. 6. Componentwise error vs. bound for Algorithm 2, for ill-conditioned cases ($\kappa_{\text{comp}} \geq 1/\gamma\varepsilon_w$).

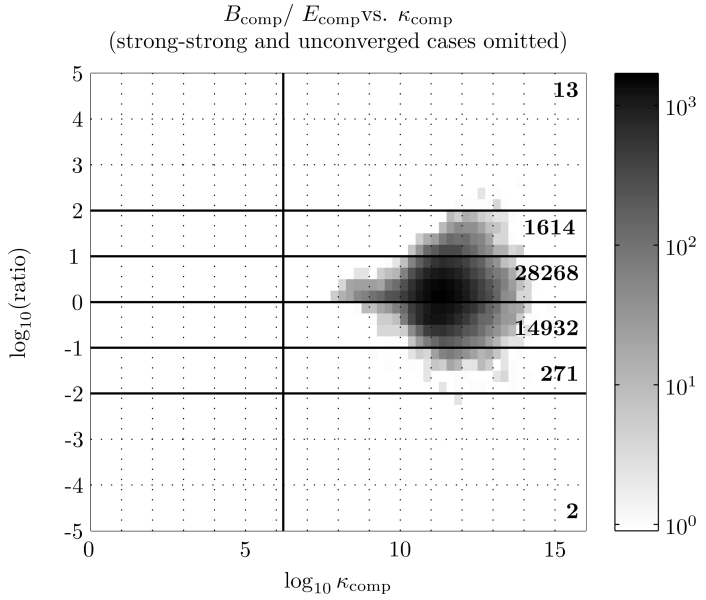


Fig. 7. Ratio $B_{\text{comp}}/E_{\text{comp}}$ vs. κ_{comp} for Algorithm 2. Cases with strong convergence (in both E_{comp} and B_{comp}) and cases with no convergence ($B_{\text{comp}} > \sqrt{\varepsilon_w}$) are omitted for clarity.

Table I. Statistics (Max, Mean, and Median) on the Number of Iterations Required by Each Algorithm (Algorithms W and L do not use doubled-x scheme so only totals are given)

	Single x			Doubled x			Total			Doubled-x
	Max	Mean	Med	Max	Mean	Med	Max	Mean	Med	Incidence
Alg. 2 (any ρ_{thresh})	3	1.5	1	3	0.7	1	4	2.1	2	55%
Alg. W (Wilkinson)							4	2.1	2	
Alg. L (LAPACK)							4	2.6	3	

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

	Single x			Doubled x			Total			Doubled-x
	Max	Mean	Med	Max	Mean	Med	Max	Mean	Med	Incidence
Alg. 2 ($\rho_{\text{thresh}} = 0.5$)	1	1	1	32	3.6	3	33	4.6	4	100%
Alg. 2 ($\rho_{\text{thresh}} = 0.8$)	1	1	1	89	4.0	3	90	5.0	4	100%
Alg. 2 ($\rho_{\text{thresh}} = 0.9$)	1	1	1	175	4.1	3	176	5.1	4	100%
Alg. 2 ($\rho_{\text{thresh}} = 0.95$)	1	1	1	330	4.3	3	331	5.3	4	100%
Alg. W (Wilkinson)							29	4.0	3	
Alg. L (LAPACK)							6	2.4	2	

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

of iterations where the solution x is represented in working (single) precision versus doubled precision. The behavior of Algorithm 2 for different values of ρ_{thresh} is also shown and will be discussed in Section 6.4.

For well-conditioned problems, both Algorithms 2 (for any value of ρ_{thresh}) and W required at most 4 iterations with a median of 2. Doubled-x iteration was triggered in 55% of cases. On average, 2/3 of the iterations are in single-x and 1/3 in the more expensive doubled-x. Algorithm L is limited to at most 6 iterations in all cases.

For ill-conditioned problems, the median number of iterations used by Algorithm 2 with $\rho_{\text{thresh}} = 0.5$ rises to 4 which is still quite modest but, in the worst case, we do 33 iterations. Doubled-x iteration is always triggered right after the first iteration.

6.4 Effects of Various Parameters in Algorithm 2

Compared to Algorithm W, Algorithm 2 incorporates several new algorithmic ingredients and adjustable parameters. We note that different parameter settings in Algorithm 2 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.

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. To measure its benefit, we ran Algorithm 2 with and without the doubled-x iteration for the two million test cases. See Demmel et al. [2004, Figure 13] for 2D histograms of E_{norm} versus B_{norm} and E_{comp} versus B_{comp} with and without doubled-x iteration.

To summarize, with doubled-x iteration, Algorithm 2 obtains 3% more cases of strong-strong normwise convergence as well as 13% more cases of

Table II. 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 "$> 100\times$")

	Underestimates		Overestimates		
	$> 100\times$	$> 10\times$	$> 100\times$	$> 10\times$	No Convergence
Alg. 2 with $\rho_{\text{thresh}} = 0.5$	0	7	1	25	40459
Alg. 2 with $\rho_{\text{thresh}} = 0.8$	0	30	3	151	25452
Alg. 2 with $\rho_{\text{thresh}} = 0.9$	0	34	3	505	22755
Alg. 2 with $\rho_{\text{thresh}} = 0.95$	0	33	14	843	21673
Alg. W (Wilkinson)	6	243	35	2130	42421
Alg. L (LAPACK)	0	0	56494	57262	1942738

(a) Normwise

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

(b) Componentwise

strong-strong componentwise convergence. The number of cases where the code reports normwise nonconvergence ($B_{\text{norm}} > \sqrt{\varepsilon_w}$) decreases by 179; cases of componentwise nonconvergence ($B_{\text{comp}} > \sqrt{\varepsilon_w}$) decreases by 5,581. Doubled-x iteration also decreases the worst normwise underestimation ratio from 1,010 to 230, and the worst componentwise underestimation ratio from 6,300 to 320.

6.4.2 *Effect of* ρ_{thresh}. In Algorithm 2, ρ_{thresh} is used to determine when to stop the iteration due to lack of progress. A larger ρ_{thresh} allows the algorithm to make progress more slowly and take more steps to converge which is useful for very ill-conditioned problems. However, a larger ρ_{thresh} may cause more severe overestimates (because of the $1 - \rho_{\text{thresh}}$ factor in the denominator of the error bound) and underestimates (since we are being more aggressive to pursue a small dx).

Table I displays the statistics of the total iteration counts for various algorithms. For well-conditioned problems, Algorithm 2 (with various ρ_{thresh}) and Algorithm W both require about the same number of steps (maximum of 4 with median of 2). For ill-conditioned problems, Algorithm W requires slightly fewer iterations than Algorithm 2 (at the cost of not converging in some cases). For Algorithm 2, it is clear that a larger ρ_{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).

Table II gives the number of overestimates and underestimates of the error bounds returned by Algorithm 2 as a function of ρ_{thresh}: The number of unconverged cases drops by nearly half as ρ_{thresh} increases from 0.5 to 0.95, at the cost of more severe overestimates and underestimates.

6.5 Cautious versus Aggressive Parameter Settings

By setting ρ_{thresh} and i_{thresh} smaller or larger, Algorithm 2 can be made cautious or aggressive. For the cautious setting, we choose $\rho_{\text{thresh}} = 0.5$ and $i_{\text{thresh}} = 10$. For the aggressive setting, we choose $\rho_{\text{thresh}} = 0.9$ and $i_{\text{thresh}} = 100$. The cautious parameter setting works reliably on all well-conditioned problems and so we use it as the default setting in the algorithm. The cautious setting also works for a large fraction of the most ill-conditioned problems, achieving strong normwise convergence in 96% of cases and strong componentwise convergence in 94% 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 at the slightly higher risk of slow convergence or extreme over/underestimates of error.

6.6 Limitations of Refinement and our Bounds

In Demmel et al. [2004, Section 7], we explore selected ill-conditioned examples in detail to see how Algorithm 2 behaves when it severely overestimates or underestimates the true error. Briefly, underestimates occur when the algorithm believes it has converged (due to small dx), but to a slightly wrong answer. Overestimates occur either because of early termination or a true error that is accidentally much smaller than $\|dx\|$ would indicate.

We also tried certain specially constructed difficult problems besides the ones described in Section 5. When faced with Rump's outrageously ill-conditioned matrices [Rump 1991] and random x, our algorithm either successfully solved the systems ($O(\varepsilon_w)$ errors and bounds) or correctly reported failure to converge.

With matrices with maximal pivot growth (such as the one in Higham [2002, 166]), our algorithm gives a correct solution up to about $n = 53$; for larger n, we fail to obtain a good solution. Even though the true condition numbers of these matrices are not huge, pivot growth during the factorization stage causes the LU factors to be highly inaccurate, and, as a result, our condition estimator overestimate, the condition number. This leads our algorithm to report that these matrices are too ill-conditioned, at the same time failing to find a good solution. However, for moderate pivot growth, our algorithm manages to clean up the solution to full accuracy.

We also tried exactly singular matrices with slightly inconsistent right-hand sides, but where LU factorization still succeeded because of roundoff. Algorithm 2 computed error bounds that were fairly close to 1, that is, exceeded our reliability threshold of $\sqrt{\varepsilon_w}$ by a large margin and so correctly indicated that the results were unreliable. We applied our algorithm to problems with exactly zero solution components. When the zeros were structural zeros, that is, a result of A^{-1} and b having appropriate zero structure, these zero components were computed exactly. These exact zero components did not impact the error bounds. When the zeros occurred from numerical cancellation, the code correctly returned an effectively infinite componentwise error bound.

7. CONCLUSIONS AND FUTURE WORK

We have presented a new variation of the extra-precise iterative refinement algorithm for the solution of linear equations. With negligible extra work, we return a bound on the maximum relative error in any solution component as well as the normwise error bound. We prove this by means of an error analysis exploiting column-scaling invariance of the algorithm. With the availability of the extended precision BLAS standard, the algorithm can be implemented portably. Based on a large number of numerical experiments, we show that, for all but the worst conditioned problems (for those with condition number less than $1/\gamma\varepsilon_w$), the algorithm converges quickly, and the corresponding error bounds (normwise and componentwise) are reliable. The algorithm also converges for a large fraction of the extremely ill-conditioned problems, 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 to store the solution x (the doubled-x iteration).

Systems like MATLAB [MathWorks, Inc.] that solve $Ax = b$ return a warning when A is nearly singular, based on a condition estimator. This estimator, like Algorithm 2, costs very little beyond the triangular factorization for medium to large n. Therefore, these systems could use 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 [Anderson et al. 1999] and ScaLAPACK [Blackford et al. 1997] linear system solvers. Additional structure in symmetric and banded systems may allow better error estimates or earlier termination. The error analysis in Section 2 needs to be extended to these systems. Algorithm 2 can also be used to improve numerical stability of the parallel sparse direct solver, SuperLU_DIST [Li and Demmel 2003], with static pivoting instead of partial pivoting.

The majority of computers contain processors with Intel's IA32 architecture [Intel Corporation 2004], and support 80-bit floating-point arithmetic in hardware (although this may be supplanted in the future by 64-bit arithmetic in SSE2). Future work will extend Algorithm 2 and its error analysis to use this kind of extended precision.

Extra-precise iterative refinement may also be applied to least squares problems, eigenvalue problems, and any other method of numerical linear algebra. Ultimately, the goal should be to achieve a guaranteed tiny error for *all* algorithms in LAPACK, as long as an appropriate condition number is sufficiently less than $1/\varepsilon_w$, and to do this for a small cost beyond the most straightforward solution (e.g., $O(n^2)$ extra on top of $O(n^3)$). This will not necessarily be possible in all cases (attaining e.g., small extra cost for narrow band problems) but this level of accuracy should be made available for the standard problems of numerical linear algebra.

Given our empirical evidence that our error bounds are guaranteed to be small when the condition number is suitably bounded, it is reasonable to ask whether it is possible to prove that this is always true for an additional cost of

only $O(n^2)$. Here is an obstacle: in Demmel et al. [2001], it was essentially shown that any condition estimator that costs asymptotically less than linear equation solving must have counterexamples, that is, matrices for which it badly misestimates the condition number. This means that any approach to guaranteed high accuracy that depends on condition estimation must have a cost proportional to linear equation solving itself. The results in Demmel et al. [2001] are not for floating-point computation and do not detract from the practical utility of our results but show that mathematically guaranteed error bounds at asymptotically negligible cost may not exist.

REFERENCES

ANDERSON, E., BAI, Z., BISCHOF, C., BLACKFORD, S., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., AND SORENSEN, D. 1999. *LAPACK Users' Guide*, Release 3.0. SIAM, Philadelphia, PA. 407 pages.

ANSI/IEEE 1985. *IEEE Standard for Binary Floating Point Arithmetic*, Std 754-1985. ANSI/IEEE, New York, NY.

BJÖRCK, Å. 1990. Iterative refinement and reliable computing. In *Reliable Numerical Computation*, M. Cox and S. Hammarling, Eds. Oxford University Press, 249–266.

BLACKFORD, L. S., CHOI, J., CLEARY, A., D'AZEVEDO, E., DEMMEL, J., DHILLON, I., DONGARRA, J., HAMMARLING, S., HENRY, G., PETITET, A., STANLEY, K., WALKER, D., AND WHALEY, R. C. 1997. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA. 325 pages.

BOWDLER, H., MARTIN, R., PETERS, G., AND WILKINSON, J. 1966. Handbook series linear algebra: Solution of real and complex systems of linear equations. *Numerische Mathematik 8*, 217–234.

DEMMEL, J., DIAMENT, B., AND MALAJOVICH, G. 2001. On the complexity of computing error bounds. *Found. Comp. Math. 1*, 101–125.

DEMMEL, J., HIDA, Y., KAHAN, W., LI, X. S., MUKHERJEE, S., AND RIEDY, E. J. 2004. Error bounds from extra precise iterative refinement. Tech. Rep. UCB/CSD-04/1344, Computer Science Division, University of California at Berkeley. (Also LAPACK Working Note 165).

DONGARRA, J., BUNCH, J. R., MOLER, C. B., AND STEWART, G. W. 1979. *LINPACK Users' Guide*. SIAM, Philadelphia, PA.

FORUM, B. T. 2002a. Basic linear algebra subprograms technical (BLAST) forum standard I. *Int. J. High Perform. Comput. Applic. 16*, 1–111.

FORUM, B. T. 2002b. Basic linear algebra subprograms technical (BLAST) forum standard II. *Int. J. High Perform. Comput. Applic. 16*, 115–199.

HIGHAM, N. J. 1987. A survey of condition number estimation for triangular matrices. *SIAM Rev. 29*, 575–596.

HIGHAM, N. J. 1988. FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Trans. Math. Soft. 14*, 4, 381–396.

HIGHAM, N. J. 1990. Experience with a matrix norm estimator. *SIAM J. Sci. Stat. Comput. 11*, 804–809.

HIGHAM, N. J. 2002. *Accuracy and Stability of Numerical Algorithms* 2nd Ed. SIAM, Philadelphia, PA.

Intel Corporation 2004. *IA-32 Intel™ Architecture Software Developer's Manual*, Volume 1: Basic Architecture. Intel Corporation.

KIELBASIŃSKI, A. 1981. Iterative refinement for linear systems in variable-precision arithmetic. *BIT 21*, 97–103.

LI, X. S. AND DEMMEL, J. W. 2003. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Soft. 29* 2 (June) 110–140.

LI, X. S., DEMMEL, J. W., BAILEY, D. H., HENRY, G., HIDA, Y., ISKANDAR, J., KAHAN, W., KANG, S. Y., KAPUR, A., MARTIN, M. C., THOMPSON, B. J., TUNG, T., AND YOO, D. J. 2002. Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Soft. 28*, 2, 152–205.

MALLOCK, R. R. M. 1933. An electrical calculating machine. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character 140*, 841 (May) 457–483.

MATHWORKS, INC. Matlab™

MOLER, C. B. 1967. Iterative refinement in floating-point. *J. ACM 14*, 2, 316–321.

NAG Ltd 2005. *NAG Fortran Library Manual, Mark 21*. NAG Ltd, Oxford, UK.

RUMP, S. 1983. Solving algebraic problems with high accuracy. In *A New Approach to Scientific Computation*, U. Kulisch and W. Miranker, Eds. Academic Press, 51–120.

RUMP, S. 1995. Verified computation of the solution of large sparse linear systems. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM) 75*, S439–S442.

RUMP, S. M. 1991. A class of arbitrarily ill conditioned floating-point matrices. *SIAM J. Matrix Analy. Appl. 12*, 4 (Oct.) 645–653.

SNYDER, J. N. 1955. On the improvement of the solutions to a set of simultaneous linear equations using the ILLIAC. *J. Math. Tables other Aids Comput. 9*, 52, 177–184.

STEWART, G. W. 1973. *Introduction to Matrix Computations*. Academic Press, New York, NY.

STRASSEN, V. 1969. Gaussian Elimination is not optimal. *Numer. Math. 13*, 354–356.

TREFETHEN, L. AND SCHREIBER, R. 1990. Average-case stability of Gaussian elimination. *SIAM J. Matrix Analy. Appl. 11*, 3, 335–360.

WILKINSON, J. H. 1948. Progress report on the Automatic Computing Engine. Report MA/17/1024, Mathematics Division, Department of Scientific and Industrial Research, National Physical Laboratory, (April) Teddington, UK.

WILKINSON, J. H. 1963. *Rounding Errors in Algebraic Processes*. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, UK. (Also published by Prentice-Hall, Englewood Cliffs, NJ, Reprinted by Dover, New York, 1994).

Received March 2005; revised September 2005; accepted September 2005