Extra-precise Iterative Refinement for Overdetermined Least Squares Problems*

James Demmel† Yozo Hida‡ Xiaoye S. Li§ E. Jason Riedy¶

May 28, 2008

Abstract

We present the algorithm, error bounds, and numerical results for extra-precise iterative refinement applied to overdetermined linear least squares (LLS) problems. We apply our linear system refinement algorithm to Björck's augmented linear system formulation of an LLS problem. Our algorithm reduces the forward normwise and componentwise errors to $\mathcal{O}(\varepsilon_w)$, where ε_w is the working precision, unless the system is too ill conditioned. In contrast to linear systems, we provide two separate error bounds for the solution x and the residual r. The refinement algorithm requires only limited use of extra precision and adds only $\mathcal{O}(mn)$ work to the $\mathcal{O}(mn^2)$ cost of QR factorization for problems of size m-by-n. The extra precision calculation is facilitated by the new extended-precision BLAS standard in a portable way, and the refinement algorithm will be included in a future release of LAPACK and can be extended to the other types of least squares problems.

1 Background

This article presents the algorithm, error bounds, and numerical results of the extra-precise iterative refinement for overdetermined least squares problem (LLS):

$$\min_x \|b - Ax\|_2 \,, \tag{1}$$

where A is of size m-by-n, and $m \geq n$.

The `xGELS` routine currently in LAPACK solves this problem using a QR factorization. There is no iterative refinement routine. We propose to add two routines: `xGELS_X` (expert driver) and `xGELS_RFSX` (iterative refinement). In most cases, users should not call `xGELS_RFSX` directly.

*This research was supported in part by the NSF Grant Nos. CCF-0444486, EIA-0122599, and CNS-0325873; the DOE Grant Nos. DE-FC02-01ER25478 and DE-FC02-06ER25786. The third author was supported in part by the Director, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under contract DE-AC03-76SF00098. The authors wish to acknowledge the contribution from Intel Corporation, Hewlett-Packard Corporation, IBM Corporation, and the NSF EIA-0303575 in making hardware and software available for the CITRIS Cluster which was used in producing these research results.

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

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

§xsli@lbl.gov, Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720.

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

Our first goal is to obtain accurate solutions for all systems up to a condition number threshold of $\mathcal{O}(1/\varepsilon_w)$ with asymptotically less work than finding the initial solution, where ε_w is the working precision. To this end, we first compute the QR factorization and obtain the initial solution in hardware-supported fast arithmetic (e.g., single or double precisions). We refine the initial solution with selective use of extra precision. The extra-precise calculations may use slower software arithmetic, *e.g.* double-double [2] when working precision already is double. Our second goal is to provide reliable error bounds for the solution vector x as well as the residual vector r.

As we will show, we can provide dependable error estimates and small errors for both x and residual $r = b - Ax$ whenever the problems are acceptably conditioned. In this paper the phrase *acceptably conditioned* will mean the appropriate condition number is no larger than the threshold $\mathsf{cond_thresh} = \frac{1}{10 \cdot \max\{10, \sqrt{m+n}\} \cdot \varepsilon_w}$, and *ill conditioned* will mean the condition number is larger than this threshold. If the problem is ill conditioned, either because the matrix itself is nearly singular or because the vector x or r is ill conditioned (which can occur even if A is acceptably conditioned), then we cannot provide any guarantees. Furthermore, we can reliably identify whether the problem is acceptably conditioned for the error bounds to be both small and trustworthy. We achieve these goals by first formulating the LLS problem as an equivalent linear system [5], then applying our linear system iterative refinement techniques that we developed previously [8].

We make the following remarks regarding the scope of this paper. Firstly, we describe the algorithms and present the numerical results only for real matrices here. But the algorithms can be extended to complex matrices straightforwardly, as we already did for the linear systems refinement procedure. Secondly, we assume A has full column rank, and the QR factorization does not involve column pivoting, although column pivoting could be easily incorporated into the algorithm. In the rank-deficient cases, we recommend using the other LAPACK routines that perform QR with column pivoting (`xGELSY`) or SVD (`xGELSS` or `xGELSD`). SVD is the most reliable method to solve the low rank LLS problem, but the development of the companion iterative refinement algorithm is still an open problem.

2 Extra-precise Iterative Refinement

In this section, we present the equivalent linear system formulation for the LLS problem, and we adapt a linear system iterative refinement procedure to refine the solution of the LLS problem.

2.1 Linear system formulation

Let $r = b - Ax$ be the residual, the LLS problem is equivalent to solving the following augmented linear system of dimension $m + n$ [3, 6]:

$$\begin{pmatrix} I_m & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}. \tag{2}$$

Then, we can apply our extra-precise iterative refinement algorithm [8] to linear system (2) to obtain more accurate vectors x and r. Algorithm 1 shows a high-level sketch of the refinement procedure.

We monitor the progress of both the solution vector x and the residual vector r to provide small infinity-norm relative normwise and componentwise errors (unless the system is too ill conditioned).

Here, the QR factorization of A and the solution for the updates in Step (2) are carried out in working precision ε_w. By "doubled arithmetic" we mean the precision used is twice the working precision, which we denote as ε_d. In our numerical experiments, we use IEEE-754 single precision as the working precision ($\varepsilon_w = 2^{-24}$) and IEEE-754 double precision in our residual calculations ($\varepsilon_d = 2^{-53}$). All the extra-precise calculations are encapsulated in the eXtended precision BLAS library (XBLAS) [15]. In the XBLAS routines, the input/output arguments are stored in the working precision, the same as the standard BLAS routines. The extra-precise accumulation is performed only internally.

Note that the extra precision ε_d we use in this paper satisfies $\varepsilon_d \leq \varepsilon_w^2$. The algorithm will still improve accuracy if ε_d is in between ε_w^2 and ε_w, except that some parameters in our stopping criteria need to be adjusted, and the accuracy results will be weaker accordingly. That is, the claim that the errors can be reduced to $\mathcal{O}(\varepsilon_w)$ for acceptably conditioned problems would be based on a cond_thresh smaller than $\frac{1}{10 \cdot \max\{10, \sqrt{m+n}\} \cdot \varepsilon_w}$. Future work remains to perform numerical experiments to quantify the effect.

Algorithm 1

Obtain the first solution $x^{(1)}$ using QR factorization of A,
and compute the first LLS residual $r^{(1)} = b - Ax^{(1)}$.

Repeat

(1) Compute the residual vectors $s^{(i)}$ and $t^{(i)}$: (**may be in doubled arithmetic**)

$$s^{(i)} = b - r^{(i)} - Ax^{(i)}$$
$$t^{(i)} = -A^T r^{(i)}$$

(2) Solve for the corrections $dr^{(i)}$ and $dx^{(i)}$ using the existing QR factorization A:

$$\begin{pmatrix} I_m & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} dr^{(i)} \\ dx^{(i)} \end{pmatrix} = \begin{pmatrix} s^{(i)} \\ t^{(i)} \end{pmatrix}$$

(3) Update the solution vectors: (**may be in doubled arithmetic**)

$$r^{(i+1)} = r^{(i)} + dr^{(i)}$$
$$x^{(i+1)} = x^{(i)} + dx^{(i)}$$

$i = i + 1$

Until $r^{(i)}$ and/or $x^{(i)}$ "accurate enough"

2.2 Extra precision

Our extra-precise iterative refinement algorithm not only computes the residual in extended precision but also may carry the solution vector $[r; x]$ in doubled precision. Therefore, in Step (1), the first residual is computed as

$$s^{(i)} = b - (r^{(i)} + r_t^{(i)}) - A(x^{(i)} + x_t^{(i)}), \tag{3}$$

where r_t and x_t are used to store the "tails" of each floating point component of r and x.

We have extended the XBLAS with a new routine to compute this portion of the residual. The existing XBLAS routine `xGEMV2_X` sufficies to compute the other portion, $t^{(i)} = A^T(r^{(i)} + r_t^{(i)})$.

2.3 Solving the augmented system

As in the case of linear system, Step (2) can be done inexpensively using the existing factorization. Assume that we have computed the QR factorization of A, i.e, $A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$, where $Q \equiv (Q_1, Q_2)$ is an orthogonal matrix of dimension m-by-m, Q_1 is of dimension m-by-n, Q_2 is of dimension m-by-$(m-n)$, and R is an upper triangular matrix of dimension n-by-n. Then, Step (2) of each iteration involves solving a linear system of the form

$$\begin{pmatrix} I_m & Q \begin{pmatrix} R \\ 0 \end{pmatrix} \\ \begin{pmatrix} R^T & 0 \end{pmatrix} Q^T & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} s \\ t \end{pmatrix}. \tag{4}$$

Letting $u = Qd \equiv Q[d_1; d_2]$, defining $c = Q^T s \equiv [c_1; c_2]$, and multiplying the top part of (4) by Q^T, we obtain the equivalent permuted triangular system

$$\begin{pmatrix} I_n & 0 & R \\ 0 & I_{m-n} & 0 \\ R^T & 0 & 0 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ v \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ t \end{pmatrix}.$$

Solving the above triangular system, we can obtain the solution of (4) from

$$c = Q^T s, \quad R^T d_1 = t, \quad d_2 = c_2, \quad Rv = c_1 - d_1, \quad u = Qd\,. \tag{5}$$

The cost is products with Q^T and Q, two triangular solves with R^T and R, and a vector-sum, totaling $O(mn)$ work per iteration.

2.4 Stopping and solution acceptance criteria

Björck and Golub [6] suggested that the iteration be terminated when the conditions (i) (for x) and (ii) (for r) below are simultaneously satisfied:

(i) $\|dx^{(i)}\|_2 \geq 0.125\|dx^{(i-1)}\|_2$ or $\|dx^{(i)}\|_2 \leq \varepsilon\|x^{(0)}\|_2$

(ii) $\|dr^{(i)}\|_2 \geq 0.125\|dr^{(i-1)}\|_2$ or $\|dr^{(i)}\|_2 \leq \varepsilon\|r^{(0)}\|_2$

This means that for both x and r, we can terminate the iteration either when it does not make enough progress (first set of inequalities), or when convergence occurs (second set of inequalities). Note that there is a small pitfall in the convergence test, because the correction $dx^{(i)}$ is compared with the first solution $x^{(0)}$ (or $r^{(0)}$), not the current solution $x^{(i)}$. If $x^{(0)}$ is completely wrong, the algorithm may converge too early.

A significant class of uses involve nearly consistent systems where the true residual r is nearly zero. Requiring convergence by measuring $\|dr\|$ relative to $\|r\|$ is too stringent. A tiny residual often will be considered ill conditioned, so we should not require excessive calculations in this fairly common case. Since $Ax = b - r$, a small $\|dr\|_2$ relative to $\|b\|_2$ indicates convergence in the range. Therefore, we propose to use the following as one of the stopping criteria:

$$\|dr^{(i)}\| \leq \varepsilon\|b\|$$

Our algorithm attempts to achieve small errors for both r and x, and in both normwise and componentwise measures. In other words, for those users who do want r as accurately as possible, we will also try to compute r with a small componentwise error. We will use the following quantities in stopping criteria:

- For normwise error, we will scale r by $1/\|b\|_\infty$, and x by $1/\|x\|_\infty$.
- For componentwise error, we will scale each r_i by $1/|r_i|$, and each x_i by $1/|x_i|$.

We use two state variables x-state and r-state ($\in$ {working, converged, no-progress}) to track normwise convergence status for x and r, and two other state variables xc-state and rc-state ($\in$ {unstable, working, converged, no-progress}) to track componentwise convergence status for x and r. Fig. 1 depicts the state transition diagrams for both normwise (Fig. 1(a)) and componentwise (Fig. 1(b)) convergence statuses.

The normwise state variable x-state (or r-state) is initialized to be in working state. The state of x is changed to converged when the correction $dx^{(i+1)}$ is too small to change the solution x^i much (condition $dx_x \stackrel{\text{def}}{=} \|dx^{(i+1)}\|/\|x^i\| \leq \varepsilon$). When convergence slows down sufficiently (condition $dxrat \stackrel{\text{def}}{=} \|dx^{(i+1)}\|/\|dx^{(i)}\| > \rho_{\text{thresh}}$), the state is changed from working to no-progress. The iteration may continue because the other state variables are still making progress, thus, the no-progress state may change back to working state when convergence becomes faster again (condition $dxrat \leq \rho_{\text{thresh}}$).

The componentwise state variable xc-state (or rc-state) is initialized to be in unstable state, meaning that each component of the solution has not settled down for convergence testing. When all the components are settled down (condition $dxc^{(i+1)} \stackrel{\text{def}}{=} \max_j |dx_j^{(i+1)}/x_j^i| \leq c_{\text{thresh}}$), the state is changed from unstable to working. c_{thresh} is the threshold below which the solution is considered stable enough for testing the componentwise convergence, and $dxcrat \stackrel{\text{def}}{=} dxc^{(i+1)}/dxc^{(i)}$. The rest of the state transitions follow the rules similar to those of the normwise case.

Finally, we terminate the refinement iteration when all four state variables (x-state, r-state, xc-state, and rc-state) are no longer in working state, or when an iteration count threshold is reached. Thus, we are simultaneously testing four sets of stopping criteria—as long as at least one of the four state variables is in working state, we continue iterating.

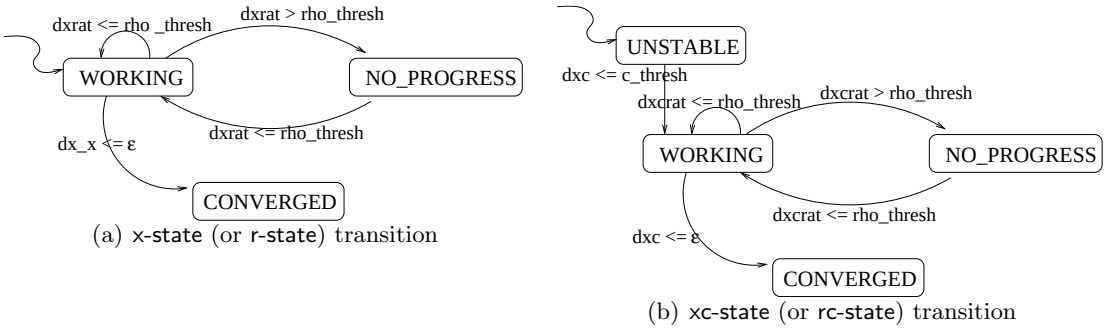
Once the algorithm has terminated, we evaluate the conditioning of each solution in both norm- and componentwise senses. If the solution is acceptably conditioned *and* the algorithm terminated in the converged state for that norm, we accept the result as accurate. Otherwise we reject the solution. Our current implementation returns an error estimate of 1.0 to signify rejected solutions. Table 1 summarizes our acceptance criteria.

In the numerical experiments presented in Section 4, we use $\rho_{\text{thresh}} = 0.5$ and $c_{\text{thresh}} = 0.25$, which we recommend to be the default settings in the code. Note that a larger ρ_{thresh} (aggressive setting) allows the algorithm to make progress more slowly and take more steps to converge, which may be useful for extremely difficult problems.

In contrast to linear systems, here we separate the augmented solution into two parts, x and r. If both converge, the algorithm should behave the same as for linear systems [8]. On the other hand, we may have half of the solution (say x) converge while the other half (r) stops making progress. Or it may be the case that half is ill conditioned while the other half is not. We will provide condition analysis of this "two-parts" approach in Section 3, and show empirically in Section 4 that the augmented systems for LLS are harder than the usual linear systems because r resides in the data space while x resides in the solution space, and they may be of different scales.

	Accepted	Rejected
condition estimate $\geq$ cond_thresh		X
solution & norm not converged		X
converged and cond. est. $<$ cond_thresh	X	

Table 1: Summary of conditions where a particular solution and norm are accepted as accurate. $\text{cond_thresh} = \frac{1}{10 \cdot \max\{10, \sqrt{m+n}\} \cdot \varepsilon_w}$.



(a) x-state (or r-state) transition

(b) xc-state (or rc-state) transition

Figure 1: State transition diagrams. ρ_{thresh} is the threshold on the ratio of successive corrections, above which the convergence is considered too slow. (a) normwise, and (b) componentwise.

2.5 Scaling

In the case of linear systems, we typically perform row and/or column equilibrations to avoid over/underflows and to reduce the condition number of scaled A. We now discuss scaling for the LLS problems.

The driver routine `xGELS_X` first uses a simple scaling before solving the LLS problem. It independently scales A by α and b by β up or down to make sure their norms are in the range $[\texttt{SMLNUM}, 1/\texttt{SMLNUM}]$, where $\texttt{SMLNUM} \approx \texttt{UNDERFLOW}/\varepsilon_w$. The LLS problem we solve becomes

$$\min_y \|\beta \cdot b - \alpha \cdot Ay\|_2 = \min_y \|\beta(b - A(\alpha/\beta \cdot y)\|_2 = \min_y \|b - A(\alpha/\beta \cdot y)\|_2 \equiv \min_x \|b - Ax\|_2 \,,$$

where $x = \alpha/\beta \cdot y$. In addition, `xLARFG`, which computes a single Householder reflection, does very careful scaling if necessary, and depends on `xNRM2` to not over/underflow.

The analysis of the linear system iterative refinement suggests that the convergence rate of the refinement procedure depends on the condition number of the augmented matrix $B = \begin{pmatrix} I_m & A \\ A^T & 0 \end{pmatrix}$. Björck suggested using a non-zero scalar α to scale the LLS problem as in $b - Ax \Rightarrow (b - Ax)/\alpha$ [3]. This does not change the computed LLS solution, but now the augmented system involves the matrix $B_\alpha = \begin{pmatrix} \alpha I_m & A \\ A^T & 0 \end{pmatrix}$. It was shown by Björck that the eigenvalues of B_α are

$$\lambda(B_\alpha) = \begin{cases} \frac{\alpha}{2} \pm (\frac{\alpha^2}{4} + \sigma_i^2)^{1/2} \\ \alpha \end{cases}$$

where $\sigma_i, i = 1, \ldots, n$ are the singular values of A. In particular, if we choose $\alpha = \sigma_{\min}/\sqrt{2}$, the condition number $\kappa_2(B_\alpha)$ takes the minimum value, $\kappa_2(B_\alpha) = \frac{1}{2} + \sqrt{\frac{1}{4} + 2\,\kappa_A^2} \leq 2\,\kappa_2(A)$, which should lead to fastest convergence of the refinement for the augmented system. However, when carefully working through the iteration in Algorithm 1 and the solve procedure (5), we see that only the quantities related to r (i.e., $t^{(i)}$, $dr^{(i)}$, and $r^{(i)}$) are scaled by $1/\alpha$; the other quantities remain the same. Thus, as long as α is chosen to be a power of the radix, the computed solution and the rounding errors are not changed, Although B_α has a lower condition number, it does not matter for our algorithm. We have confirmed this from numerical experiments. From the analysis in Section 3 we will see that $\kappa(B_\alpha)$ does not play any role in the conditioning of the LLS problem.

Now, we discuss general equilibrations. We cannot row equilibrate A in our framework as this would change its range and the norm we are minimizing. We can potentially column equilibrate A by a general diagonal matrix D, which means scaling x by D^{-1}. As long as we pick the entries in D to be powers of the radix, we would not introduce any rounding errors (modulo over/underflows). The normwise convergence criteria for x would be affected, because $\|dx\|$, $\|x\|$, and $\|dx^{(i+1)}/\|dx^{(i)}\|$ will be different for unscaled or scaled x. With $\mathcal{O}(n)$ work, we can compute these quantities corresponding to the unscaled x, and use them in the stopping criteria, just as we did for linear systems [8]. It remains our future work to see how this type of scaling affects the convergence rate.

3 Normwise and Componentwise Condition Numbers

The refinement algorithm can return various error bounds for the solutions x and r. In order to determine when our error bounds are reliable, we need condition numbers for both x and r.

Applying the standard componentwise perturbation analysis to (2), we assume that the computed solutions $\tilde{r} = r + \delta r$ and $\tilde{x} = x + \delta x$ satisfy the equations

$$\begin{pmatrix} I_m & A + \delta A \\ (A + \delta A)^T & 0 \end{pmatrix} \begin{pmatrix} \tilde{r} \\ \tilde{x} \end{pmatrix} = \begin{pmatrix} b + \delta b \\ 0 \end{pmatrix}, \tag{6}$$

where $|\delta A| \leq \varepsilon |A|$, and $|\delta b| \leq \varepsilon |b|$. This is equivalent to the perturbed normal equation

$$(A + \delta A)^T (A + \delta A)\tilde{x} = (A + \delta A)^T (b + \delta b).$$

Since $A^T A x = A^T b$, subtracting it from the above equation and noting that $\tilde{r} = b + \delta b - (A + \delta A)\tilde{x}$, we get

$$\delta x = A^+ (\delta b - \delta A \cdot \tilde{x}) + (A^T A)^{-1} \delta A^T \cdot \tilde{r}\,, \tag{7}$$

where $A^+ = (A^T A)^{-1} A^T$. Using the inequalities $|\delta A| \leq \varepsilon |A|$ and $|\delta b| \leq \varepsilon |b|$, we obtain the following bound on each component of δx

$$|\delta x| \leq \varepsilon \left[|A^+| \left(|b| + |A| \cdot |\tilde{x}| \right) + |(A^T A)^{-1}| \cdot |A^T| \cdot |\tilde{r}| \right].$$

Taking norms, we get

$$\frac{\|\delta x\|_\infty}{\|\tilde{x}\|_\infty} \leq \varepsilon \frac{\left\| |A^+| (|b| + |A| \cdot |\tilde{x}|) \right\|_\infty + \left\| |(A^T A)^{-1}| \cdot |A^T| \cdot |\tilde{r}| \right\|_\infty}{\|\tilde{x}\|_\infty}\,.$$

Therefore, the following can be used as a normwise condition number for x:

$$\kappa^x_{norm} \stackrel{\text{def}}{=} \frac{\left\| |A^+|(|b| + |A| \cdot |\tilde{x}|) \right\|_\infty + \left\| |(A^T A)^{-1}| \cdot |A^T| \cdot |\tilde{r}| \right\|_\infty}{\|\tilde{x}\|_\infty} . \tag{8}$$

Note that this derivation is valid assuming that $A + \delta A$ has full rank, i.e., $\kappa_2(A) \equiv \|A\|_2 \|A^+\|_2$ is not too large. In Section 4.2, we will confirm this assumption in practice.

In the case of linear systems, the basic solution method (GEPP) and the iterative refinement algorithm are column scaling invariant. Therefore, we may choose the column scale factors such that each component of the scaled solution is of magnitude about one; thus, the usual normwise error bound measures the componentwise error as well.

We would like to apply the same technique to the augmented linear system (2). Letting $D_r \approx diag(r)$, and $D_x \approx diag(x)$, our refinement iterations can be thought of as solving the following scaled system

$$\begin{pmatrix} I_m & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} D_r & \\ & D_x \end{pmatrix} \begin{pmatrix} z_r \\ z_x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} \tag{9}$$

where $z_r = D_r^{-1} r \approx \mathbf{1}$, and $z_x = D_x^{-1} x \approx \mathbf{1}$. The same solution method for the augmented system can be used, but the convergence of z_r and z_x are monitored simultaneously as with r and x.

Applying the analogous perturbation analysis to (9), we assume that the computed solutions $\tilde{z_r} = z_r + \delta z_r$ and $\tilde{z_x} = z_x + \delta z_x$ satisfy the equations

$$\begin{pmatrix} I_m & A + \delta A \\ (A + \delta A)^T & 0 \end{pmatrix} \begin{pmatrix} D_r & \\ & D_x \end{pmatrix} \begin{pmatrix} \tilde{z_r} \\ \tilde{z_x} \end{pmatrix} = \begin{pmatrix} b + \delta b \\ 0 \end{pmatrix} . \tag{10}$$

By the same algebraic manipulation, we obtain an expression for δz_x that is simply the expression for δx (7) with a pre-factor D_x^{-1}. That is,

$$\delta z_x = D_x^{-1} \left[A^+(\delta b - \delta A \tilde{x}) + (A^T A)^{-1} \delta A^T \tilde{r} \right]$$

Taking norms, we get

$$\|\delta z_x\|_\infty \leq \varepsilon \left(\left\| |D_x^{-1}||A^+|(|b| + |A| \cdot |\tilde{x}|) \right\|_\infty + \left\| |D_x^{-1}| \cdot |(A^T A)^{-1}| \cdot |A^T| \cdot |\tilde{r}| \right\|_\infty \right) \tag{11}$$

Since $z_x \approx \mathbf{1}$, $\|\delta z_x\|_\infty \approx \max_i \frac{|\delta z_x|_i}{|z_x|_i} = \max_i \frac{|D_x^{-1} \delta x|_i}{|D_x^{-1} x|_i} = \max_i \frac{|\delta x|_i}{|x|_i}$. Therefore, (11) measures the componentwise error of x, and the following can be used as the componentwise condition number:

$$\kappa^x_{comp} \stackrel{\text{def}}{=} \left\| |D_x^{-1}| \cdot |A^+|(|b| + |A| \cdot |\tilde{x}|) \right\|_\infty + \left\| |D_x^{-1}| \cdot |(A^T A)^{-1}| \cdot |A^T| \cdot |\tilde{r}| \right\|_\infty . \tag{12}$$

We can use LAPACK's condition estimator (`SLACON`, based on Hager-Higham's algorithm [11]) to estimate each norm in (12). For example, let $d = |b| + |A||\tilde{x}|$, $D = diag(d) > \mathbf{0}$, and e be the vector of all ones; then,

$$\begin{aligned} \left\| |D_x^{-1}| \cdot |A^+|(|b| + |A| \cdot |\tilde{x}|) \right\|_\infty &= \left\| |D_x^{-1}| \cdot |A^+| \, d \right\|_\infty = \left\| |D_x^{-1}| \cdot |A^+| \, De \right\|_\infty = \left\| |D_x^{-1}| \cdot |A^+| \, D \right\|_\infty \\ &= \left\| |D_x^{-1} A^+ D| \right\|_\infty = \left\| D_x^{-1} A^+ D \right\|_\infty \end{aligned}$$

The condition estimator requires multiplying $D_x^{-1} A^+ D$ and $(D_x^{-1} A^+ D)^T$ with some vectors, which involves triangular solves using R if we have $A = QR$ at hand.

Similarly, let $d = |A^T| \cdot |\tilde{r}|$, $D = diag(d) > \mathbf{0}$; then,

$$\left\| |D_x^{-1}| \cdot |(A^T A)^{-1}| \cdot |A^T| \cdot |\tilde{r}| \right\|_\infty = \left\| |D_x^{-1}| \cdot |(A^T A)^{-1}| \, D \right\|_\infty = \left\| D_x^{-1} (A^T A)^{-1} D \right\|_\infty ,$$

which can be estimated analogously.

We now derive the condition numbers for r. Consider the first set of equations in (6):

$$r + \delta r = b + \delta b - (A + \delta A)\tilde{x}\,.$$

Subtracting $r = b - Ax$ from the above equation, we have $\delta r = \delta b - \delta A\tilde{x} - A\delta x$. Substituting (7) for δx, we obtain

$$\begin{aligned} \delta r &= \delta b - \delta A\tilde{x} - \left(AA^{+}(\delta b - \delta A\tilde{x}) + A(A^TA)^{-1}\delta A^T\tilde{r}\right) \\ &= (I_m - AA^{+})(\delta b - \delta A\tilde{x}) - (A^{+})^T\delta A^T\tilde{r} \end{aligned} \tag{13}$$

Then, the bounds for each component of δr and δz_r are given by

$$|\delta r| \le \varepsilon\left[|I_m - AA^{+}|\,(|b| + |A|\cdot|\tilde{x}|) + |(A^{+})^T|\cdot|A^T|\cdot|\tilde{r}|\right],$$

$$|\delta z_r| \le \varepsilon|D_r^{-1}|\left[|I_m - AA^{+}|(|b| + |A|\cdot|\tilde{x}|) + |(A^{+})^T|\cdot|A^T|\cdot|\tilde{r}|\right].$$

Thus, the normwise condition number for r can be defined as

$$\kappa^r_{norm} \stackrel{\text{def}}{=} \frac{\|\,|I_m - AA^{+}|\,(|b| + |A|\cdot|\tilde{x}|)\,\|_\infty + \left\||(A^{+})^T|\cdot|A^T|\cdot|\tilde{r}|\right\|_\infty}{\|b\|_\infty}\,. \tag{14}$$

The componentwise condition number for r can be defined as

$$\kappa^r_{comp} = \left\||D_r^{-1}|\cdot|I_m - AA^{+}|(|b| + |A|\cdot|\tilde{x}|)\right\|_\infty + \left\||D_r^{-1}|\cdot|(A^{+})^T|\cdot|A^T|\cdot|\tilde{r}|\right\|_\infty\,. \tag{15}$$

Observe that, given $A = (Q_1, Q_2)\begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1R$, we have $I_m - AA^{+} = I_m - Q_1Q_1^T$. Since $\|I_m - Q_1Q_1^T\|_2 = \min\{1, m-n\}$, we expect the effect of multiplication by $|I_m - AA^{+}|$ to be small, and so we can use the following approximate but cheaper-to-compute formula as the normwise condition number for r:

$$\kappa^r_{norm_cheap} \stackrel{\text{def}}{=} \frac{\|\,|b| + |A|\cdot|\tilde{x}|\,\|_\infty + \left\||(A^{+})^T|\cdot|A^T|\cdot|\tilde{r}|\right\|_\infty}{\|b\|_\infty}\,. \tag{16}$$

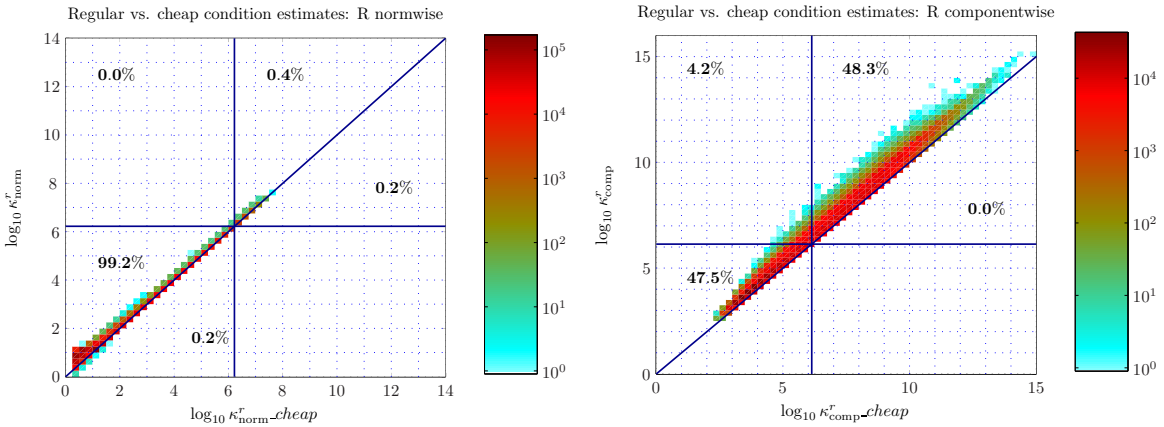
By the same token, the following cheap formula could be considered as the componentwise condition number for r:

$$\kappa^r_{comp_cheap} = \left\||D_r^{-1}|\cdot(|b| + |A|\cdot|\tilde{x}|)\right\|_\infty + \left\||D_r^{-1}|\cdot|(A^{+})^T|\cdot|A^T|\cdot|\tilde{r}|\right\|_\infty\,. \tag{17}$$

Again, Hager-Higham's algorithm can be used to estimate the respective norms $\left\||(A^{+})^T|\cdot|A^T|\cdot|\tilde{r}|\right\|_\infty$ and $\left\||D_r^{-1}|\cdot|(A^{+})^T|\cdot|A^T|\cdot|\tilde{r}|\right\|_\infty$.

Examining Eq. (12) for κ^x_{comp}, we notice that the second term can be proportional to $\kappa(A)^2$, depending on the size of residual r. That is, when the LLS problem is compatible or nearly so (with small r), the second term is negligible, and the sensitivity of x to perturbation is similar to the case of linear system. When the angle between b and Ax is large (with large r), or closer to $\pi/2$, the solution x would have small components, which are much more sensitive to perturbation.

Fig. 3 compares the condition numbers estimated from formulae (14)–(17). There are altogether one million randomly generated test matrices of size 100×50 (see Section 4.1 for tests generation).



(a) normwise (b) componentwise

Figure 2: Comparison between regular and cheap condition estimations for R.

Fig. 2(a) shows that in normwise measure, the two estimates given by (14) and (16) are very close—they agree to within a factor four, with the cheaper ones yielding slightly smaller values. Therefore, it is safe for our code to use (16). On the other hand, in the componentwise measure (see Fig. 2(b)), the cheap estimation given by (17) can be much smaller than the one given by (15)—sometimes over 400x smaller. This is because if D_r has highly varying entries, leaving out the $|I_m - AA^+|$ factor could make the estimation arbitrarily small. In this case, we shall use the more expensive formula (15).

Table 2 summarizes the estimated condition numbers and error bounds to be returned from the new LAPACK routine. Here,

- γ is a constant factor used for setting a threshold to differentiate the acceptably conditioned problems from the ill conditioned ones. Based on our empirical results from the linear system refinement, a good value of γ is $\max\{10, \sqrt{m+n}\}$ [8].
- ε_w is the machine epsilon in working precision.
- $\rho_{\max}$ is the maximum ratio of successive corrections, which is a conservative estimate of the rate of convergence. For example, $\rho^x_{\max} \stackrel{\text{def}}{=} \max_{j \le i} \frac{\|dx^{(j+1)}\|_\infty}{\|dx^{(j)}\|_\infty}$. Similarly, $\hat{\rho}_{\max}$ is defined for the componentwise case.

The code also returns a componentwise backward error $BERR(\tilde{r}, \tilde{x}) = \max(\omega_1, \omega_2)$ [5, pp. 36], where

$$\omega_1 = \max_{1 \le i \le m} \frac{|\tilde{r} + A\tilde{x} - b|_i}{(|\tilde{r}| + |A||\tilde{x}| + |b|)_i}, \quad \omega_2 = \max_{1 \le i \le n} \frac{|A^T \tilde{r}|_i}{(|A|^T |\tilde{r}|)_i}.$$

4 Numerical Experiments

We now present numerical results from our iterative refinement algorithm. All the test runs were conducted on the CITRIS Itanium-2 Cluster at UC Berkeley.[1] Each node has dual 1.3 GHz Itanium

[1] `http://www.millennium.berkeley.edu/citris/`

		Condition number	Relative error	Error estimate
x	norm.	κ^x_{norm} (8)	$E^x_{norm} = \frac{\|\tilde{x}-x\|_\infty}{\|\tilde{x}\|_\infty}$	$B^x_{norm} = \max\left\{\frac{\|dx^{(i+1)}\|_\infty/\|x^i\|_\infty}{1-\rho^x_{\max}}, \gamma\varepsilon_w\right\}$
	comp.	κ^x_{comp} (12)	$E^x_{comp} = \max_k \left\|\frac{\tilde{x}(k)-x(k)}{\tilde{x}(k)}\right\|$	$B^x_{comp} = \max\left\{\frac{\|D_x dx^i\|_\infty}{1-\hat{\rho}^x_{\max}}, \gamma\varepsilon_w\right\}$
r	norm.	κ^r_{norm} (16)	$E^r_{norm} = \frac{\|\tilde{r}-r\|_\infty}{\|b\|_\infty}$	$B^r_{norm} = \max\left\{\frac{\|dr^{(i+1)}\|_\infty/\|b^i\|_\infty}{1-\rho^r_{\max}}, \gamma\varepsilon_w\right\}$
	comp.	κ^r_{comp} (15)	$E^r_{comp} = \max_k \left\|\frac{\tilde{r}(k)-r(k)}{\tilde{r}(k)}\right\|$	$B^r_{comp} = \max\left\{\frac{\|D_r dr^i\|_\infty}{1-\hat{\rho}^r_{\max}}, \gamma\varepsilon_w\right\}$

Table 2: Summary of condition numbers and error estimates, where $D_x = diag(x)$ and $D_r = diag(r)$. The error estimates are derived from the analysis of the geometric-decreasing sequence of errors from iterations, see [8].

2 (Madison) processors. The operating system is Linux 2.6.18, and the compilers and flags are `ifort -no-ftz -fno-alias -funroll-loops -O3` for Fortran code, and `icc -O3` for C code.

We use single precision $\varepsilon_w = 2^{-24} \approx 6 \times 10^{-8}$ as working precision and double precision $\varepsilon_d = 2^{-53} \approx 10^{-16}$ in the refinement. Running in single precision allows us to test the input space more thoroughly. Also, we can rely on the well-tested double precision codes to produce trustworthy true solutions. The iteration control parameters ρ_{thresh} and c_{thresh} are set to 0.5 and 0.25, respectively. Recall that $\gamma = \max\{10, \sqrt{m+n}\}$. We will analyze in detail the results of one million least squares problems of dimension 100×50.

Recall that we would like to achieve the following two goals from the refinement procedure:

Goal 1 For acceptably conditioned problems, we obtain small errors.

Goal 2 A small error bound returned from the code is trustworthy.

We will demonstrate that we meet these goals with the following data:

- norm- and componentwise relative errors in x (Section 4.2.1), where the errors are $\leq \gamma \cdot \varepsilon_w$ whenever the specific condition number $\leq \frac{1}{10\,\gamma\,\varepsilon_w}$;
- error estimates for x (Section 4.2.2), where the error estimates are within a factor of 10 of the true error whenever the specific condition number $\leq \frac{1}{10\,\gamma\,\varepsilon_w}$;
- norm- and componentwise relative errors in r (Section 4.3.1), similar to x;
- error estimates for r (Section 4.3.2), similar to x; and
- iteration counts (Section 4.4), where the median number of iterations for acceptably conditioned systems is two.

4.1 Test problem generation

We need to generate many test cases with a wide range of condition numbers, and the right-hand sides b having angles ranging from close to zero to close to $\pi/2$ with respect to the column span of A. We now describe our procedure for generating the test problems.

- Generate matrix A.

1. Randomly pick a condition number κ with $\log_2 \kappa$ distributed uniformly in $[0, 24]$. 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 $\varepsilon_w = 2^{-24}$).
2. 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

$$A = U\Sigma \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix}, \tag{18}$$

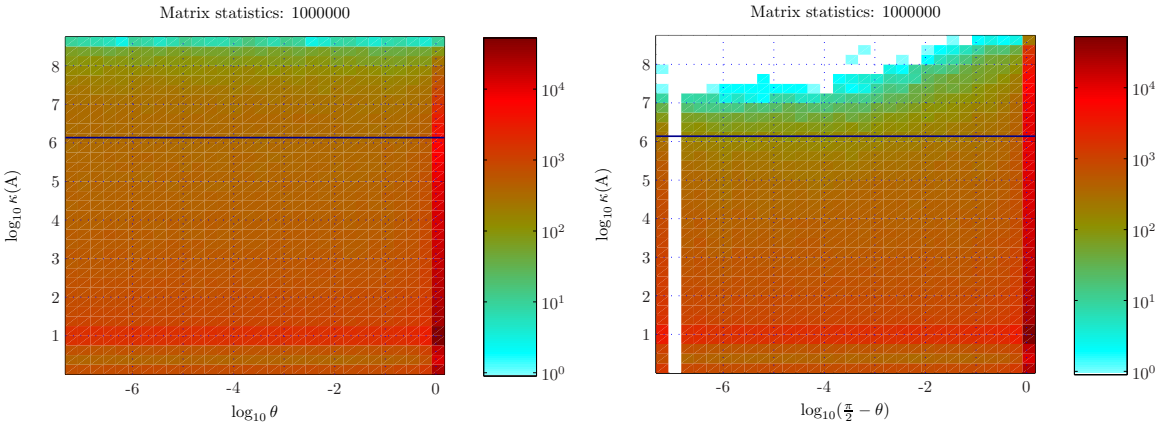
   where U, V_1, and V_2 are random orthogonal matrices with dimensions n, k, and $n - k$, respectively. If κ is large, this makes the leading k columns of A nearly linearly dependent. U, V_1 and V_2 are applied via sequences of random reflections of dimensions 2 through n, k or $n - k$, respectively.

- Generate vector b.
  1. Compute $b_1 = Ax$ (with random x) using double-double precision but rounded to single precision in the end (using the XBLAS routine `BLAS_sgemm_x`). Normalize b_1 such that $\|b_1\|_2 = 1$.
  2. Compute $A = QR$. Pick vector d of m random real numbers uniformly distributed in $(-1, 1)$. Compute $b_2 = d - QQ^T d$, and normalize b_2 such that $\|b_2\|_2 = 1$. Now b_2 is orthogonal to the columns of A.
  3. Pick a random $\theta \in [0, \pi/2]$ as follows. First set $\theta = \pi \cdot 2^u$, where u is uniformly distributed in $[-26, -1]$. Then, with probability 0.5, flip $\theta = \pi/2 - \theta$. Compute final $b = \cos\theta \cdot b_1 + \sin\theta \cdot b_2$.

  Note that steps 2 and 3 are carried out in double precision. The final b is then rounded to single precision.

- Compute x_{true} and r_{true} by solving the augmented linear equations (2) using double precision linear system solver with double-double precision iteration refinement (the new LAPACK routine `DGESVXX`).

Using the above procedure, we have systematically generated one million test problems of dimension 100×50. Among them, the condition numbers of A range between 1.0 and 4.9×10^8, and the condition numbers of the scaled augmented matrix B_α (see Section 2.5) range between 57.8 and 8.1×10^9. Fig. 3 shows the 2D histograms of the properties of the test problems: the angle θ between b and the column span of A, and the conditioning of A. In the 2D histogram, each shaded square at coordinate (x, y) indicates the population of the problems that have $\theta \in [10^x, 10^{x+1/4}]$ and condition number $\kappa_\infty(A) \in [10^y, 10^{y+1/4}]$. Fig. 3(a) plots against θ, while Fig. 3(b) plots against



(a) θ (b) $\frac{\pi}{2} - \theta$

Figure 3: Statistics of the one million test problems.

$\pi/2 - \theta$. These plots show that we indeed have a good sampling of easy and hard problems, having many cases with θ close to 0 and many cases with θ close to $\pi/2$. In fact, we sample the region around tiny and near-$\pi/2$ angles much more than the "easier" middle region. They also show that our method generated angles arbitrarily close to zero, but not arbitrarily close to $\pi/2$.

These types of 2D histogram plots will be used in the remainder of the paper to illustrate all the test results.

4.2 Results for x

Of the one million test problems, 577,412 are acceptably conditioned in normwise measure (i.e., $\kappa^x_{norm} < \frac{1}{10\,\gamma\,\varepsilon_w}$), and 355,330 are acceptably conditioned in componentwise measure (i.e., $\kappa^x_{comp} < \frac{1}{10\,\gamma\,\varepsilon_w}$).

4.2.1 True error

First, in Fig. 4, we plot the normwise true error of x obtained from single precision QR factorization without iterative refinement (the LAPACK routine `SGELS`). As shown, the error grows proportionally with the condition number of x, κ^x_{norm}. The horizontal solid line is at the value $E^x_{norm} = \gamma\varepsilon_w \approx 7.3 \cdot 10^{-7}$. A matrix below this line indicates that the true error is of order ε_w, which is the most desirable convergence case. We also plot this solid line in the other analogous figures for all the errors (e.g., E^x_{comp}, E^r_{norm}, and E^r_{comp}) and the error bounds (e.g., $B^x_{norm}, B^x_{comp}, B^r_{norm}$, and B^r_{comp}). The vertical solid line (also in the other analogous figures) is at the value $\kappa^x_{norm} = \frac{1}{10\,\gamma\,\varepsilon_w} \approx 1.4 \cdot 10^5$, which separates the acceptably conditioned problems from the ill conditioned ones. In addition, we often plot a dashed line at the value of ε_w as a reference line. In each quadrant defined by the horizontal and vertical cutoff lines, we label the number of test problems falling in that quadrant.

Fig. 5 shows the results of normwise true error of x after our refinement procedure terminates, whether the code reports convergence or not. Regarding κ^x_{norm}, there are 577,412 acceptably conditioned problems that appear in the left two quadrants. Among them, 577,377 converged, and

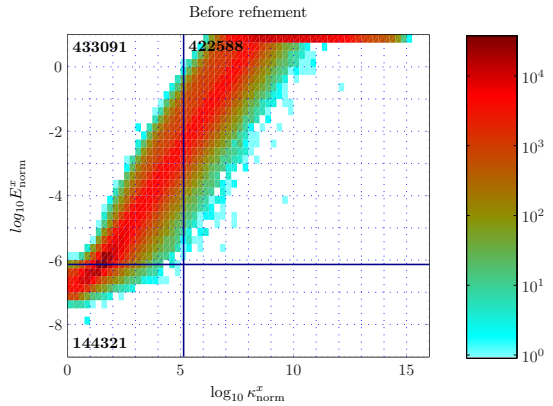


Figure 4: Before refinement: true error of x vs. κ^x_{norm}.

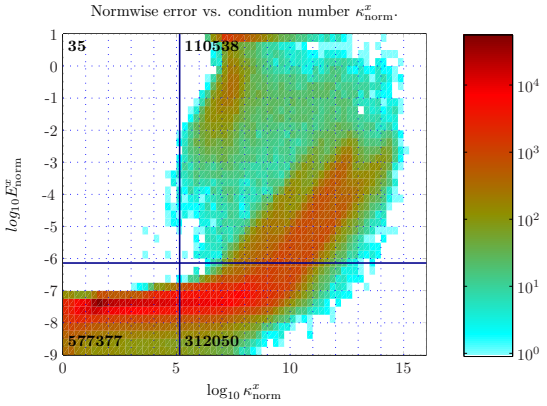


Figure 5: After Refinement: true error of x vs. κ^x_{norm} (*all cases*).

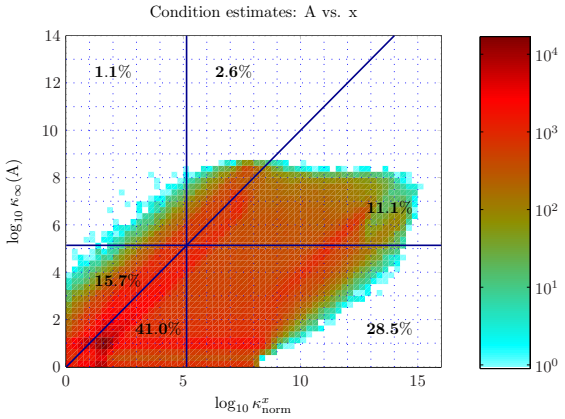


Figure 6: $\kappa_\infty(A)$ vs. κ^x_{norm}.

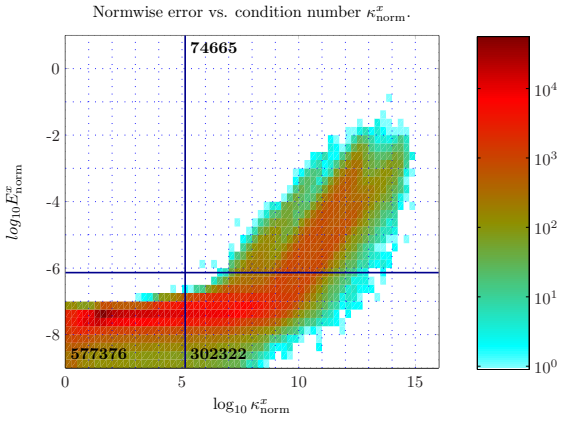


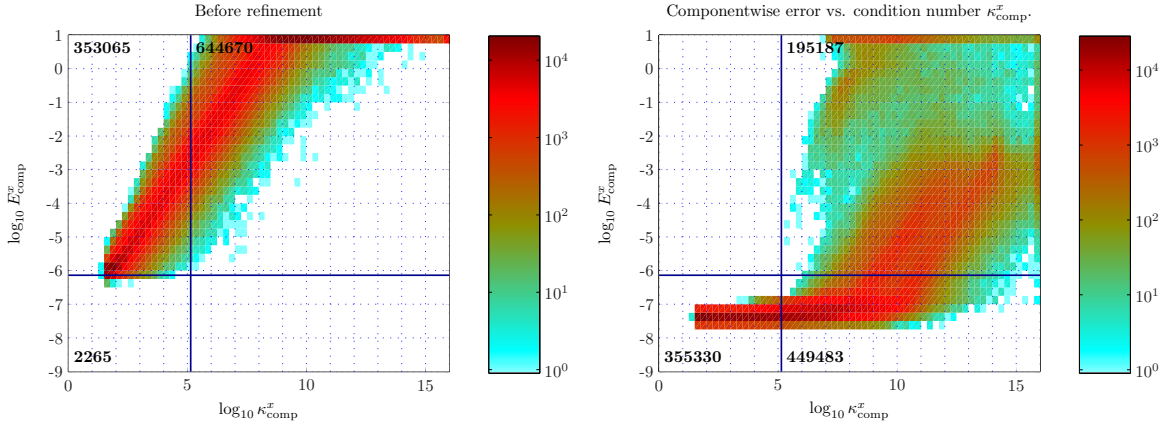
Figure 7: Normwise error of x vs. κ^x_{norm} (*converged cases*).

35 have errors larger than our error threshold. For those 35 problems, the condition number of A ($\kappa_\infty(A) \stackrel{\text{def}}{=} \|A\|_\infty \|A^+\|_\infty$) is larger than or close to the condition threshold; thus they are close to rank deficient. Fig. 6 compares the condition number of A vs. the condition number of x. As shown, many problems have $\kappa_\infty(A)$ larger than κ^x_{norm}.

In the situation when A is nearly rank deficient, our computed κ^x_{norm} may not be trustworthy for two reasons: 1) our perturbation analysis for Eq. (8) assumes that A is not too ill conditioned, and 2) κ^x_{norm} depends on the computed $\tilde{x}$ and $\tilde{r}$, which are not accurate. Therefore, these problems should also be characterized as ill conditioned. In fact, with closer examination of those 35 cases, we found that all of them did not converge. Our code only accepts the results when the iteration converges w.r.t. the corresponding state variable. When it does not converge, we simply set the error estimate to one. In Fig. 7, we make another histogram plot that filters out the unconverged cases. That is, we show only the problems with x-state = converged (see Section 2.4). Now, none of the problems appears in the upper left quadrant.

Fig. 8 shows the results of componentwise true error of x versus condition number. There are

355,330 acceptably conditioned problems. Without refinement (Fig. 8(a)), the error grows proportionally with the condition number. After refinement (Fig. 8(b)), all the acceptably conditioned problems converged. Among the remaining very ill conditioned ones, 449,483 of them still converged. Overall, only 19.5% (195,187 out of one million) of the problems did not converge. Fig. 9 plots only the cases that have converged.



(a) before refinement (b) after refinement

Figure 8: Componentwise error of x vs. condition number (*all cases*).

From Figs. 7 and 9 we draw the most important conclusion that for all the acceptably conditioned and converged problems, our algorithm delivers tiny errors for x, both normwise and componentwise. That is, **Goal 1** is clearly achieved for the solution x. Moreover, the algorithm converged with small errors for a large fraction of the ill conditioned problems, that is, 73.8% of the 422,588 problems with $\kappa^x_{norm} \geq \frac{1}{10\,\gamma\,\varepsilon_w}$, and 69.7% of the 644,670 problems with $\kappa^x_{comp} \geq \frac{1}{10\,\gamma\,\varepsilon_w}$.

4.2.2 Reliability of error bounds

From the user's point of view, a small error can be recognized only from a small error bound estimated from the algorithm. Figs. 10 and 11 show the relation between the true errors and the error bounds (see Table 2). In these plots, the diagonal line marks where the true error equals the error bound. A matrix above the diagonal corresponds to an overestimate, and a matrix below the diagonal corresponds to an underestimate.

For the acceptably conditioned (measured in κ^x_{norm} and κ^x_{comp}, respectively) and converged problems (Figs. 10(a) and 10(b)), the algorithm converged to a solution with error of at most $1.1 \cdot 10^{-7}$, and returned an error bound $\gamma\varepsilon_w \approx 7.3 \cdot 10^{-7}$, which never underestimates the true error.

For the ill conditioned problems (Figs. 11(a) and 11(b)), a large fraction of the problems still converged and the error bounds reflect small errors. Note that in cases where both B^x_{norm} and E^x_{norm} converged to below $\gamma\varepsilon_w$ (we call it *strong-strong* convergence), or completely fails to converge, the code would return a small error bound or flag the failure with an error bound of 1.0. Thus, an interesting scenario is when the algorithm claims convergence to a solution but the error estimation is far from the true solution. We analyze this situation in Fig. 12, where we plot the 2D histogram of the ratio B^x_{norm}/E^x_{norm} (resp. B^x_{comp}/E^x_{comp}) against κ^x_{norm} (resp. κ^x_{comp}). In the normwise case (Fig. 12(a)), there are 74,812 matrices falling in this category, which accounts for 17.7% of the total very ill conditioned ones. The error estimations become worse as the condition numbers of

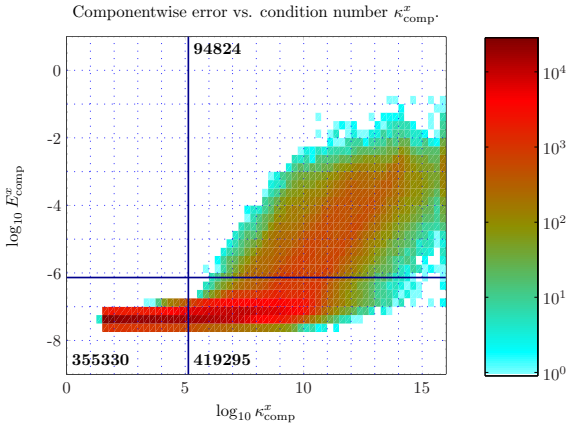
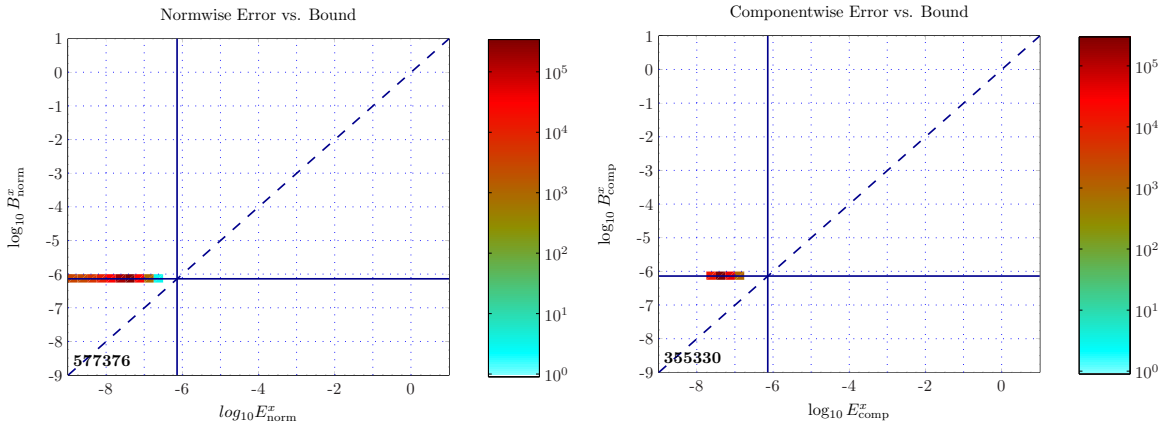
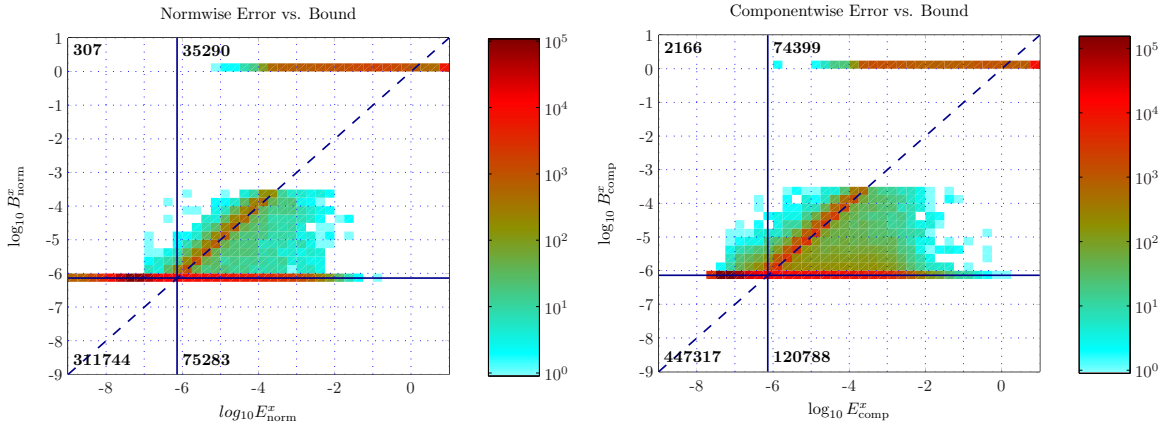


Figure 9: Componentwise error of x vs. κ^x_{comp} (*converged cases*).



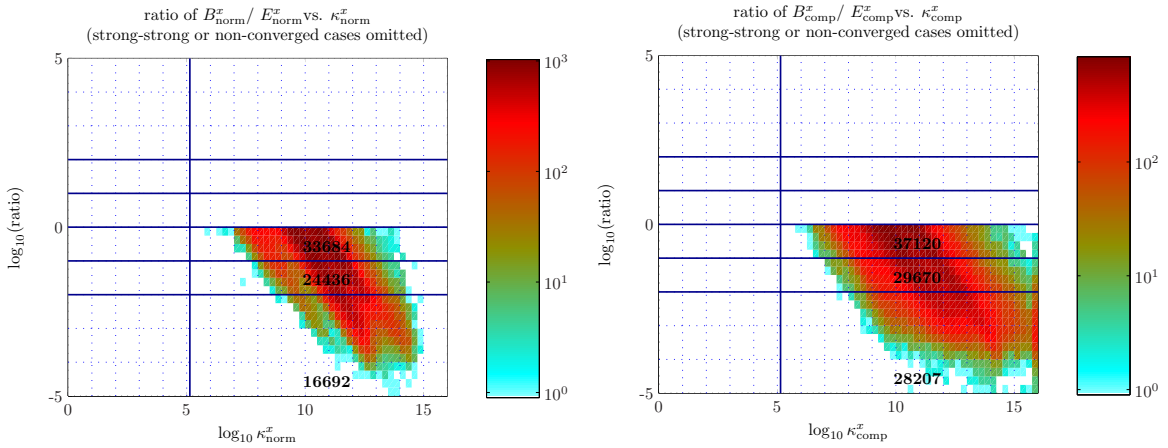
(a) normwise (b) componentwise

Figure 10: Error estimate of x versus true error (*acceptably conditioned and converged cases*).



(a) normwise (b) componentwise

Figure 11: Error estimate of x versus true error (*ill conditioned or non-converged cases*).



(a) normwise (b) componentwise

Figure 12: Overestimation and underestimation ratio of bound over error for x (*strong-strong converged or non-converged cases are omitted*).

the problems get larger. This is because the code converged to the wrong solutions, and the error bounds are not trustworthy. Similar behavior is seen in the componentwise case (Fig. 12(b)). Here, 94,997 matrices fall in this category, accounting for 14.7% of the total very ill conditioned ones.

These data show that **Goal 2** is achieved for acceptably conditioned problems, because the returned error bounds are close to and never underestimate the true errors. For very ill conditioned problems, the error bounds may underestimate the true errors, depending on how difficult the problems are. In any event, because the code detects ill conditioning, the returned error bound can be set to one to warn the user.

4.3 Results for r

Of the one million test problems, 962,834 are acceptably conditioned in normwise measure (i.e., $\kappa^r_{norm} < \frac{1}{10\,\gamma\,\varepsilon_w}$), and 412,683 are acceptably conditioned in componentwise measure (i.e., $\kappa^r_{comp} < \frac{1}{10\,\gamma\,\varepsilon_w}$).

4.3.1 True error

Figs. 13 and 14 show the relation between the normwise error and the condition number κ^r_{norm} before and after refinement, respectively. There are 962,834 acceptably conditioned problems that appear in the left two quadrants. Among them, 958,102 converged, and 4,732 have errors larger than our error threshold. The condition numbers of A for those 4,732 problems are all larger than the condition threshold; thus, similar to the situation with x, we should classify them as ill conditioned.

In Fig. 15, we make a histogram plot that filters out the unconverged cases. That is, we show only the problems with r-state = converged (see Section 2.4). Now, none of the cases appears in the upper left quadrant, and the algorithm delivers small errors for all converged problems.

Fig. 16 shows the error vs. the angle between b and Ax for those acceptably conditioned problems ($\kappa^r_{norm} < \frac{1}{10\,\gamma\,\varepsilon_w}$) but unconverged (r-state $\neq$ converged, see Section 2.4). These include all

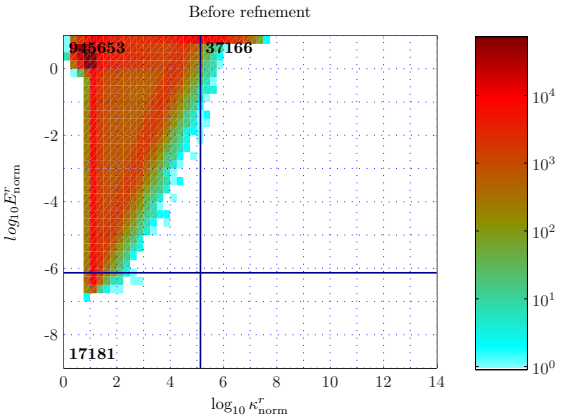


Figure 13: Before refinement: normwise error of r vs. κ^r_{norm}.

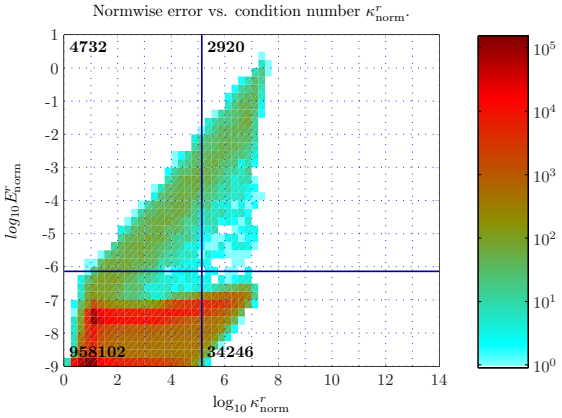


Figure 14: After refinement: normwise error of r vs. κ^r_{norm} (*all cases*).

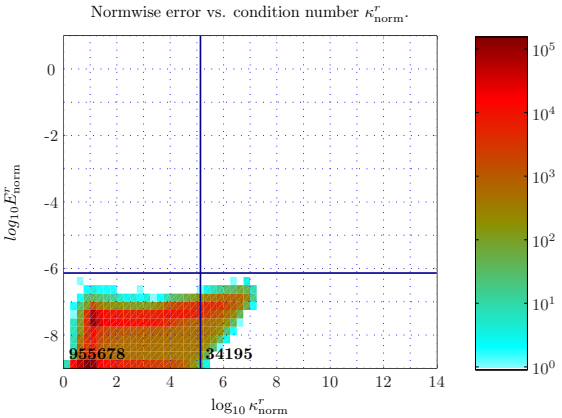


Figure 15: Normwise error of r vs. κ^r_{norm} (*converged cases*).

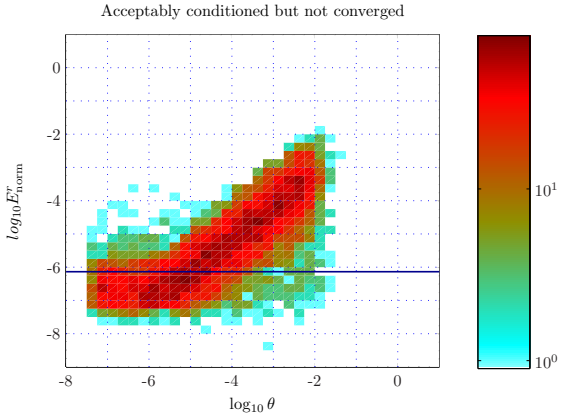


Figure 16: Normwise error of r vs. θ (*acceptably conditioned but unconverged*).

the problems appearing in the upper left quadrant of Fig. 14. This shows that the error grows proportionally with the angle θ, although θ is still small.

Fig. 17 compares the condition number of A vs. the condition number of r. It can be seen that many problems have $\kappa_\infty(A)$ larger than κ^r_{norm}. Fig. 18 shows the relation of the condition ratio of r over A with the size of the angle θ.

In Fig. 19, we plot the errors with respect to the maximum of the two condition numbers. In this measure, all the acceptably conditioned problems converged. Among the remaining very ill conditioned ones, 144,257 of them still converged with small errors. Overall, only 0.7% (7,652 out of one million) of the problems did not converge.

Fig. 20 plots the 2D histogram of the true error of r vs. κ^r_{norm}, in which $\|r\|_\infty$ instead of $\|b\|_\infty$ is used in the denominator when computing the relative error and κ^r_{norm} (see Eq. (14)). In this case, all but one of the acceptably conditioned problems converged. On the other hand, many more problems now become ill conditioned (cf. Fig. 14). In particular, most (nearly) consistent systems (with small r) would be considered ill conditioned in this measure. We believe this is too sensitive

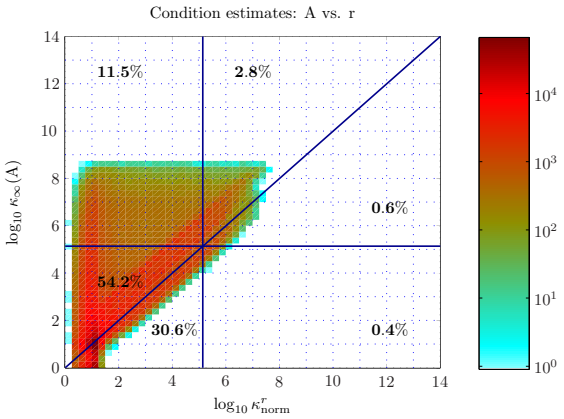


Figure 17: $\kappa_\infty(A)$ vs. κ^r_{norm} (*all cases*).

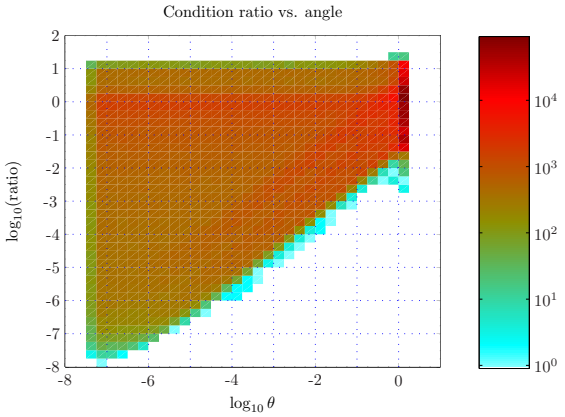


Figure 18: Ratio of $\frac{\kappa^r_{norm}}{\kappa_\infty(A)}$ vs. θ (*all cases*).

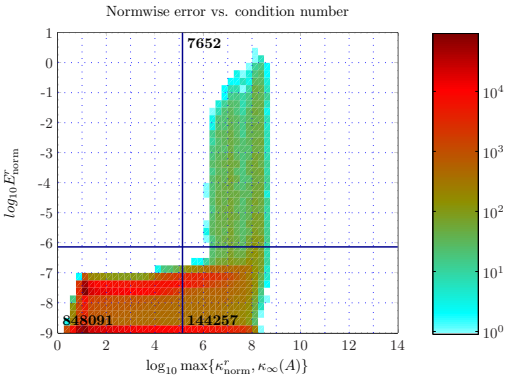


Figure 19: Normwise error of r vs. $\max\{\kappa^r_{norm}, \kappa_\infty(A)\}$ (*all cases*).

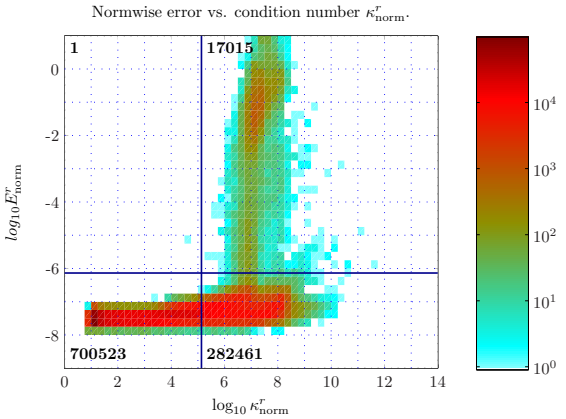
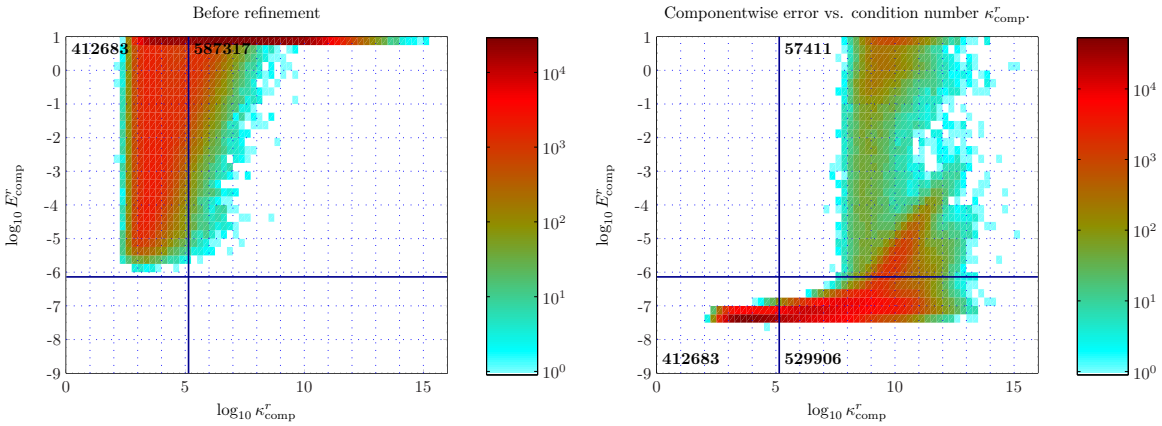


Figure 20: Normwise error of r vs. condition number (*all cases*). κ^r_{norm} is measured w.r.t $\|r\|_\infty$ instead of $\|b\|_\infty$ in formula (14).



(a) before refinement (b) after refinement

Figure 21: Componentwise error of r vs. κ^r_{comp} (*all cases*).

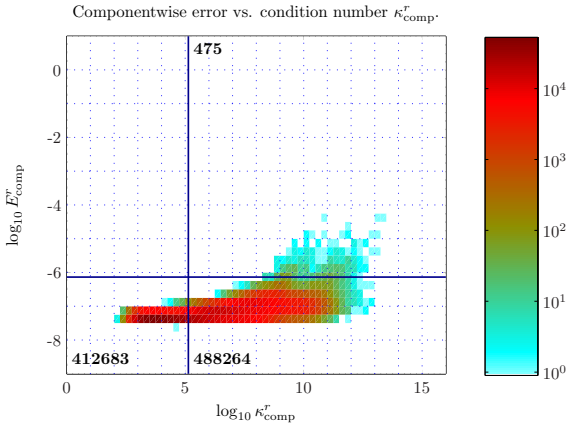


Figure 22: Componentwise error of r vs. κ^r_{comp} (*converged cases*).

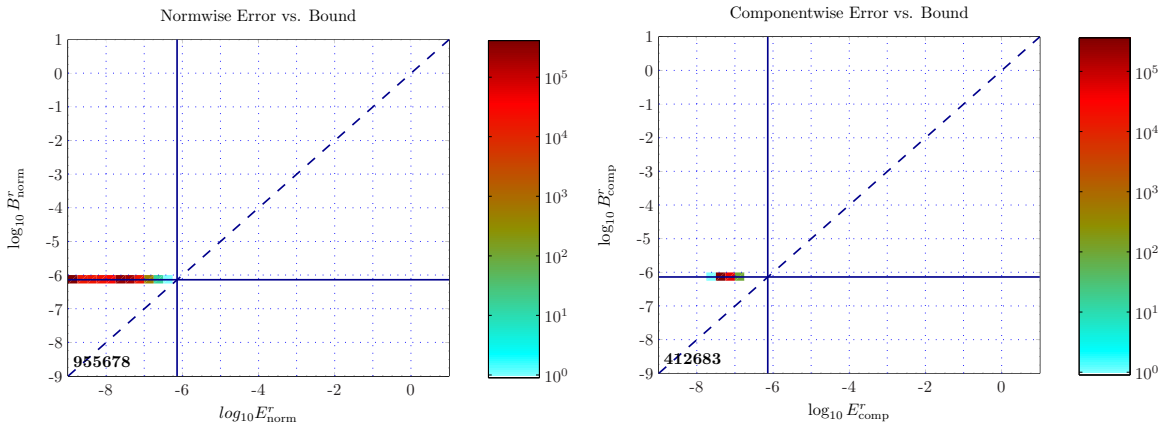
a measure of ill conditioning for many users of least squares problems, who only want to know that Ax is near b.

In componentwise measure, the algorithm delivers tiny errors for all the acceptably conditioned problems after refinement (Fig. 21(b)). That is, **Goal 1** is met for r in componentwise measure. Moreover, the algorithm converged with small errors for a large fraction of the very ill conditioned problems. Overall, only 5.7% (57,411 out of one million) did not converge.

In Fig. 22, we make a histogram plot that filters out the unconverged cases. That is, we show only the problems with rc-state = converged (see Section 2.4).

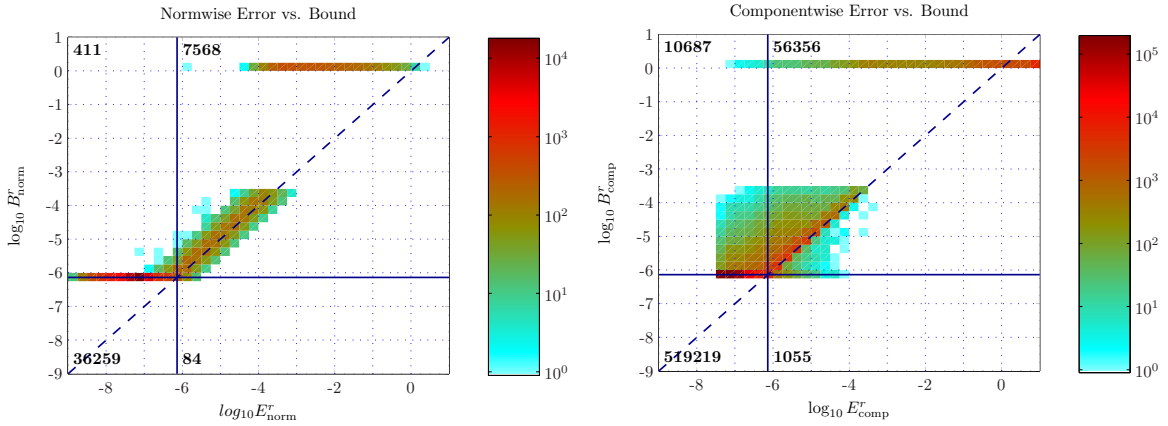
4.3.2 Reliability of error bounds

Figs. 23 and 24 show the relation between the true errors and the error bounds (see Table 2). For the acceptably conditioned (measured in κ^r_{norm} and κ^r_{comp}) and converged problems, both normwise and componentwise (Fig. 23), the algorithm converged to a solution with true error of at most $1.1 \cdot 10^{-7}$,



(a) normwise (b) componentwise

Figure 23: Error estimate of r versus true error (*acceptably conditioned and converged cases*).



(a) normwise (b) componentwise

Figure 24: Error estimate of r versus true error (*ill conditioned or non-converged cases*).

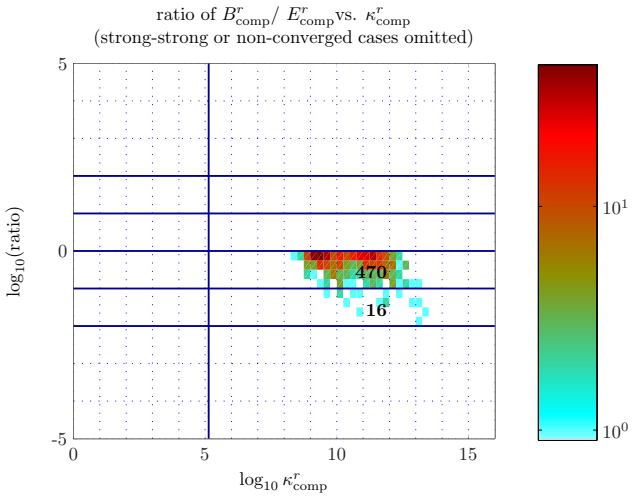


Figure 25: Overestimation and underestimation ratio of bound over error for r (*strong-strong converged or non-converged cases are omitted*).

and returned an error bound $\gamma\varepsilon_w \approx 7.3 \cdot 10^{-7}$, which never underestimates the true error. This shows that **Goal 2** is met for acceptably conditioned problems.

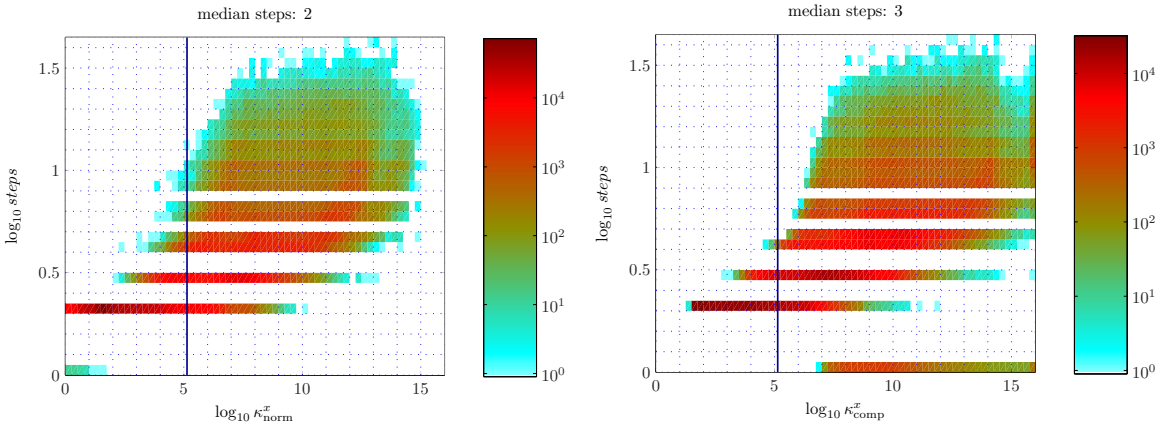
For the ill conditioned problems (Figs. 24(a) and 24(b)), a large fraction of the problems still converged and the error bounds reflect small errors. Similar to x, we are more concerned with the scenario when the algorithm claims convergence to a solution but the error estimate is far from the true solution. In the normwise case, none of the problems falls in this category. In the componentwise case, only 486 problems fall in this category. Fig. 25 plots the 2D histogram of the ratio B^r_{comp}/E^r_{comp} against κ^r_{comp}. The error bounds correctly estimate the errors (within a factor of 10) for a large fraction of the problems, and only for 16 of them, the error bounds underestimate by more than a factor of 10, yet no more than a factor of 100.

4.4 Iteration count

Fig. 26 shows the profile of the number of iterations plotted against the four condition numbers κ^x_{norm}, κ^x_{comp}, κ^r_{norm}, and κ^r_{comp}. Each sub-figure shows only the number of steps the algorithm took for one specific quantity to be either converged or making no more progress. For example, Fig. 26(a) shows the number of steps the convergence status variable x-state was working before it converged or made no-progress, and the horizontal axis shows the corresponding condition number. The four convergence status variables are defined for x and r, normwise and componentwise, see Section 2.4. For acceptably conditioned problems w.r.t. the corresponding condition number, the algorithm usually terminates very quickly. For ill conditioned problems, it may take many more steps—sometimes up to 43. The median number of steps to terminate in our tests is three.

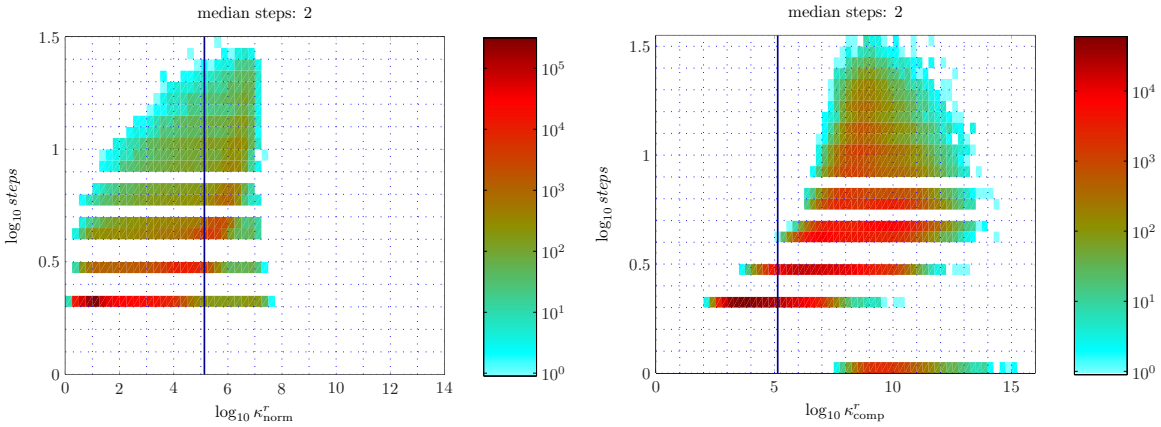
Fig. 27 is a summary plot showing the number of steps for the iteration to stop entirely after all four measures are either converged or making no progress. The horizontal axis is the maximum of all four condition numbers. Table 3 summarizes the median and max number of iteration steps with individual metrics.

We also compared the iteration count between Björck and Golub's algorithm (see Section 2.4) and our algorithm using $\rho_{\text{thresh}} = 0.125$. With this ρ_{thresh}, the two algorithms measure progress



(a) steps to terminate x-state vs. κ^x_{norm}

(b) steps to terminate xc-state vs. κ^x_{comp}



(c) steps to terminate r-state vs. κ^r_{norm}

(d) steps to terminate rc-state vs. κ^r_{comp}

Figure 26: Iteration count w.r.t. individual metrics.

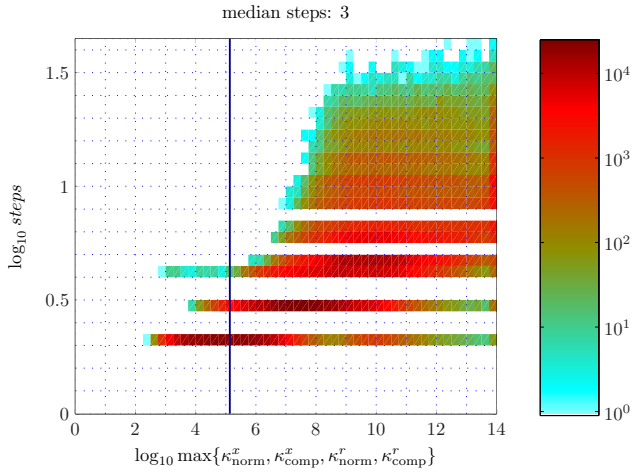


Figure 27: Steps to terminate the entire iteration vs. the maximum of the four condition numbers.

		Acceptably conditioned		All systems	
		median	max	median	max
x	norm.	2	11	2	43
	comp.	2	4	3	43
r	norm.	2	28	2	31
	comp.	2	3	2	37

Table 3: Summary of iteration counts.

similarly, but Björck and Golub's algorithm considers convergence relative to the *initial* solution and instead of the current iterate. Also, Björck and Golub's algorithm only considers normwise convergence of x and r, whereas our algorithm considers both normwise and componentwise convergence. Our stronger termination criteria sometimes require more iterations, but the median across our test set for both algorithms is three iterations. The occasional extra work buys reliable componentwise results. Surprisingly, there are also cases where Björck and Golub's algorithm requires more iterations. Carrying x and r to extra precision allows our algorithm to terminate more quickly in approximately 1% of our test cases.

5 Special Cases

A truly robust code should be able to handle the special cases such as when A is square or A is block diagonal. We now discuss how we may adapt the code in these situations.

Given a square system, the algorithm designed for LLS may return unexpected results depending on the rank of the matrix. If the matrix is of full rank, the true residual is a constant zero regardless of the right-hand side. The condition number of a constant function is zero, and the code should compute a zero residual and accept it. However, if the matrix is rank deficient or within a rounding error of rank deficiency, then the least-squares problem is ill-posed without additional constraints. The code should detect the low-rank or nearly low-rank matrices and reject the solutions.

If we evaluate κ^r_{norm} with formula (14) for square A, the computed condition number is usually small ($\mathcal{O}(\varepsilon)$), but not exactly zero, because $\tilde{r}$ is the residual of the perturbed system and should be exactly zero or small, and the evaluated $I_m - AA^+$ is usually $\mathcal{O}(\varepsilon)$ instead of 0. Therefore, it does not accurately return the true condition numbers reflecting both cases (1) and (2). Similarly, evaluating formula (15) for κ^r_{comp}, we would get zero-divided-by-zero error from zero diagonals of D_r.

It appears that for square matrices, we could simply call our linear system solver `xGESVXX` [8], which uses LU factorization. But then we cannot return Q and R for later use. A better strategy is as follows. We will have a new routine, say `xGEQRSVXX`, which runs with the same algorithm and stopping criteria as `xGESVXX`, but uses QR in place of LU as the basic solution method and for computing the updates. As long as x converges successfully, we can return r and its condition numbers identically zero. If `xGEQRSVXX` discovers the system is too ill conditioned, we should return a large number as r's condition numbers.

When matrix A is block diagonal and all the blocks are acceptably conditioned and either square or overdetermined, our algorithm can obtain as good an answer as if we were solving an individual LLS problem with each block. If some block is low rank, as long as the corresponding block in the right-hand side is zero, our algorithm will return zero for that block of x, and return good

answers for the other blocks. In the more general rank-deficient cases, we recommend using the other LAPACK routines that perform QR with column pivoting or SVD. The development of the companion iterative refinement routines remains future work.

6 Related Work

In an earlier paper [21], Wampler compared twenty different computer programs to assess numerical accuracy of five different algorithms. Wampler found that those programs using orthogonal Householder transformations, classical Gram-Schmidt orthonormalization or modified Gram-Schmidt orthogonalization were generally much more accurate than those using elimination algorithms, and the most successful programs accumulated inner products in double precision and made use of iterative refinement procedures. This justifies that we start from the algorithm by Björck & Golub [6].

Much of the analysis in Section 3 appeared previously in the literature, see for example [4, 5, 12]. Our main contribution is to provide a unified framework for both normwise and componentwise condition numbers and error bounds that are easy to understand and can be easily computed. More recently, Cucker et al. have provided sharper bounds for condition numbers of the least squares problems [7]. However, those quantities cannot be easily estimated, and hence we do not use them in this work. Arioli et al. [1] considered perturbations that are measured in different norms (Frobenius or the spectral norm for A and the Euclidean norm for b), and derived the condition number for x expressed in the weighted product norms. Although their condition numbers can be cheaply estimated, their norms are different from the infinity-norm we are using. The refinement algorithms for weighted and linearly contrained least squares also exist [10, 17].

Motivated by the speed difference between single and double precision floating point arithmetics on many recent microprocessors (single precision typically can be 2x faster, and is 10x faster on the IBM Cell Processor), Langou et al. have performed similar studies with iterative refinement for linear systems, where the LU factorization is computed in single precision, and the iterative refinement is performed in double precision [14]. Here, iterative refinement is used to accelerate the solution of $Ax = b$, rather than getting a more accurate answer. The stopping criterion in [14] is simply that the residual satisfies

$$\|Ax - b\|_2 \leq \sqrt{n}\varepsilon_w \|A\|_F \|x\|_2 \tag{19}$$

This guarantees that on convergence, their algorithm computes an $\hat{x}$ with a small normwise backward error (i.e., satisfying $(A + \delta A)\hat{x} = b + \delta b$ with $\|\delta A\|_2 = O(\varepsilon_w)\|A\|_2$ and $\|\delta b\|_2 = O(\varepsilon_w)\|b\|_2$), which is the same error bound obtained by Gaussian elimination with pivoting done in *double* precision without refinement. They showed that on ten different architectures, this new mixed precision routine is faster (up to 1.9x on the Cell) than the routine that computes LU in double precision and does not perform refinement. This is because refinement does only $O(n^2)$ double precision operations as opposed to the $\frac{2}{3}n^3 + O(n^2)$ single precision operations of Gaussian elimination,

In Section 7 we show which backward error could be used as a stopping criterion to mimic this behavior for least squares problems.

7 Using Iterative Refinement To Solve Least Squares Problems With Small Backward Error

It would be beneficial to have a scheme that does QR decomposition in single precision plus iterative refinement in double precision to guarantee the same backward error as provided by QR decomposition in double precision with no refinement. We explain how to do this, since it is not as straightforward as for solving $Ax = b$.

With $Ax = b$ the exact backward error is well-known to be computable exactly and cheaply from the residual $\hat{r} = b - A\hat{x}$, as shown by the theorems of Rigal-Gaches [18, 12] and Oettli-Prager [16, 12]:

$$\begin{aligned} \omega_{E,f}(\hat{x}) &\equiv \min\{\eta : (A+\delta A)\hat{x} = b + \delta b,\ |\delta A| \le \eta \cdot E,\ |\delta b| \le \eta \cdot f\} \\ &= \max_i \frac{|A\hat{x} - b|_i}{E|\hat{x}| + f}, \end{aligned} \tag{20}$$

for any matrix E and vectors f and $\hat{x}$. But this is not the case for least squares. Instead, Walden, Karlson and Sun showed that the exact backward error is given by a formula requiring the smallest singular value of the m-by-$(m+n)$ matrix $[A, c(I - \hat{r}\hat{r}^T/\|\hat{r}\|^2)]$, where c is a constant [20, 12]. This is too expensive (even exploiting the special structure of the matrix) to use as a stopping criterion for iterative refinement. Instead, we consider the backward error of the (scaled) augmented system. Of course a general perturbation to the augmented system will not preserve its special structure and so not guarantee a small backward error in the original least squares problem. Fortunately, it is possible to both preserve this structure and compute the backward error inexpensively.

Theorem 1 *Let* $\omega = \max(\frac{\|\alpha\hat{r} + A\hat{x} - b\|_\infty}{\|A\|_\infty\|\hat{x}\|_1 + \|b\|_\infty}, \frac{\|A^T\hat{r}\|_\infty}{\|A\|_\infty\|\hat{r}\|_1})$, $\hat{r} = b - A\hat{x}$. *If* $\omega \ll 1$, *then* $\hat{x}$ *is a solution of the perturbed least squares problem*

$$\begin{bmatrix} \alpha I & A + \delta A \\ (A+\delta A)^T & 0 \end{bmatrix} \cdot \begin{bmatrix} \hat{\hat{r}} \\ \hat{x} \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{21}$$

with

$$\|\delta A\|_\infty \le (\sqrt{mn((m+1)^2+1)} \cdot \omega + O(\omega^2))\|A\|_\infty \ , \ \textit{and}$$
$$\|\hat{r} - \hat{\hat{r}}\|_\infty \le \frac{\omega}{1-\omega}\|\hat{r}\|_\infty \ .$$

Proof: Let $1^{r\times c}$ ($0^{r\times c}$) denote the r-by-c matrix of all ones (all zeros, resp.). Apply Oettli-Prager (20) to the scaled augmented system

$$\begin{bmatrix} \alpha I & A \\ A^T & 0^{n\times n} \end{bmatrix} \cdot \begin{bmatrix} \hat{r} \\ \hat{x} \end{bmatrix} = \begin{bmatrix} b \\ 0^{n\times 1} \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$

with

$$E = \begin{bmatrix} 0^{m\times m} & \|A\|_\infty 1^{m\times n} \\ \|A\|_\infty 1^{n\times m} & 0^{n\times n} \end{bmatrix} \quad \text{and} \quad f = \begin{bmatrix} \|b\|_\infty 1^{m\times 1} \\ 0^{n\times 1} \end{bmatrix}$$

yielding

$$\omega \equiv \omega_{E,f}([\hat{r};\hat{x}]) = \max(\frac{\|e_1\|_\infty}{\|A\|_\infty\|\hat{x}\|_1 + \|b\|_\infty}, \frac{\|e_2\|_\infty}{\|A\|_\infty\|\hat{r}\|_1}) \ .$$

In other words, there are δA_1, δA_2 and δb satisfying

$$\begin{bmatrix} \alpha I & A+\delta A_1 \\ (A+\delta A_2)^T & 0^{n\times n} \end{bmatrix} \cdot \begin{bmatrix} \hat{r} \\ \hat{x} \end{bmatrix} = \begin{bmatrix} b+\delta b \\ 0^{n\times 1} \end{bmatrix} \tag{22}$$

and

$$|\delta A_1|_{ij} \le \omega \cdot \|A\|_\infty \ , \ |\delta A_2|_{ij} \le \omega \cdot \|A\|_\infty \text{ and } |\delta b|_i \le \omega \cdot \|b\|_\infty \ .$$

But we have not yet shown that $\omega = O(\epsilon)$ is enough to guarantee a tiny backward error in the least squares problem, because we must still show that it is possible to choose $\delta A_1 = \delta A_2$. We do this in two steps. First we note that there is a matrix Δ satisfying $(I+\Delta)b = b + \delta b$ and $\|\Delta\|_\infty = \|\delta b\|_\infty / \|b\|_\infty \le \omega$, namely $\Delta = \|b\|_\infty^{-1} \cdot \delta b \cdot e_j^t$ where $\|b\|_\infty = |b_j|$ and e_j is the j-th column of the identity matrix. We apply this to (22) yielding

$$\begin{aligned}
\begin{bmatrix} b \\ 0^{n\times 1} \end{bmatrix} &= \begin{bmatrix} (I+\Delta)^{-1} & 0^{m\times n} \\ 0^{n\times m} & I \end{bmatrix} \cdot \begin{bmatrix} b+\delta b \\ 0^{n\times 1} \end{bmatrix} \\
&= \begin{bmatrix} (I+\Delta)^{-1} & 0^{m\times n} \\ 0^{n\times m} & I \end{bmatrix} \cdot \begin{bmatrix} \alpha I & A+\delta A_1 \\ (A+\delta A_2)^T & 0^{n\times n} \end{bmatrix} \cdot \begin{bmatrix} \hat{r} \\ \hat{x} \end{bmatrix} \\
&= \begin{bmatrix} \alpha I & (I+\Delta)^{-1}(A+\delta A_1) \\ (A+\delta A_2)^T(I+\Delta) & 0 \end{bmatrix} \cdot \begin{bmatrix} (I+\Delta)^{-1}\hat{r} \\ \hat{x} \end{bmatrix} \\
&= \begin{bmatrix} \alpha I & A+\delta \hat{A}_1 \\ (A+\delta \hat{A}_2)^T & 0 \end{bmatrix} \cdot \begin{bmatrix} \hat{\hat{r}} \\ \hat{x} \end{bmatrix}
\end{aligned}$$

where

$$\begin{aligned}
\|\delta \hat{A}_1\|_\infty &= \|(I+\Delta)^{-1}(A+\delta A_1) - A\|_\infty \le \frac{\omega(1+2\omega)}{1-\omega}\|A\|_\infty \\
\|\delta \hat{A}_2\|_\infty &= \|(I+\Delta^T)\delta A_2 + \Delta^T A\|_\infty \le ((m+1)\omega + m\omega^2)\|A\|_\infty \\
\|\hat{r} - \hat{\hat{r}}\|_\infty &= \|(I+\Delta)^{-1}\hat{r} - \hat{r}\|_\infty \le \frac{\omega}{1-\omega}\|\hat{r}\|_\infty
\end{aligned}$$

Now we apply a result of Kielbasinski and Schwetlick [13, 12] or of Stewart [19] that lets us replace $\delta \hat{A}_1$ and $\delta \hat{A}_2$ by a single δA of slightly larger norm, namely

$$\begin{aligned}
\|\delta A\|_F^2 &\le \|\delta \hat{A}_1\|_F^2 + \|\delta \hat{A}_2\|_F^2 \\
&\le m(\|\delta \hat{A}_1\|_\infty^2 + \|\delta \hat{A}_2\|_\infty^2) \\
&\le m((\frac{\omega(1+2\omega)}{1-\omega})^2 + ((m+1)\omega + m^2\omega^2)^2)\|A\|_\infty^2 \\
&= [m(1+(m+1)^2)\omega^2 + O(\omega^3)]\|A\|_\infty^2
\end{aligned}$$

satisfying (21). Now use $\|\delta A\|_\infty \le \sqrt{n} \cdot \|\delta A\|_F$ to get the final bound. The result of Kielbasinski and Schwetlick is invariant to the scaling of the residual, so it needs no change to accommodate the αI instead of I in the upper left corner. $\square$

8 Conclusions and Future Work

We have presented a new variant of the extra-precise iterative refinement algorithm for linear least squares problems. The algorithm can be implemented portably and efficiently using the extended

precision BLAS standard, and is readily parallelizable. The overhead of refinement and condition estimations is small when the problems are relatively large, for example, when they do not fit in the processor's cache.

Our extensive numerical tests showed that, using extra precision in residual computation and solution vector update, the algorithm achieved condition-independent accuracy for both x and r, both normwise and componentwise, as long as the problem is not more ill conditioned than $1/\varepsilon_w$. In contrast to linear systems, where conditioning of A alone usually indicates the difficulty of the problems, for least squares problems ill conditioning may be due to several factors: ill conditioning of A itself, a small angle between b and Ax (i.e., a nearly consistent system resulting in small r), or an angle close to $\pi/2$ (i.e., x is small). When the algorithm does not converge, it correctly flags these difficult situations. Measured by iteration count and the amount of computation per step, this refinement procedure requires more work than for the linear systems.

In this paper, we used rather conservative settings of the parameters ρ_{thresh} and c_{thresh} to control the iteration. If desired in very difficult cases, the user can change the default settings to make the algorithm more aggressive, and let the code progress more slowly with more steps to reach convergence.

Estimating the relevant condition numbers is an $O(mn)$ task but adds significant performance overhead to small systems as well as tall and skinny systems. To reduce condition estimation's performance impact, we are investigating using the initial step size to determine if the system is too ill conditioned. This method originally was proposed by Forsythe and Moler [9] for determining the condition number in $Ax = b$ but was abandoned because higher-precision residual calculations were not portable. Using their method to determine if a converged solution is dependable and also extending their method to componentwise accuracy and partitioned solutions ($[x; r]$) looks promising.

We believe our iterative refinement framework is applicable to the other least squares problems that can be formulated as the augmented systems. These include the linear equality-constrainted least squares problem (see the LAPACK routine `xGGLSE`), and the weighted linear least squares problem (see the LAPACK routine `xGGGLM`). The development of the corresponging iterative refinement algorithms remains to be future work.

Acknowledgement

We thank David Vu and Meghana Vishvanath of U.C. Berkeley. Their work on debugging the iterative refinement routines for linear systems helped us debug the least squares code.

We are very fortunate to have received detailed feedbacks from the referees, which led to the revised paper in a much better form.

References

[1] Mario Arioli, Marc Baboulin, and Serge Gratton. A partial condition number for linear least squares problems. *SIAM J. Matrix Analysis and Applications*, 29(2):413–433, 2007.

[2] David Bailey. A Fortran-90 double-double precision library. `http://crd.lbl.gov/~dhbailey/mpdist`.

[3] Åke Björck. Iterative refinement of linear least squares solutions I. *BIT*, 7:257–278, 1967.

[4] Åke Björck. Component-wise perturbation analysis and error bounds for linear least squares solutions. *BIT*, 31:238–244, 1991.

[5] Åke Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[6] Åke Björck and Gene Golub. Iterative refinement of linear least squares solution by Householder transformation (Algol Programming). *BIT*, 7:322–337, 1967.

[7] Felipe Cucker, Huaian Diao, and Yimin Wei. On mixed and componentwise condition numbers for Moore-Penrose inverse and linear least squares problems. *Mathematics of Computation*, 76(258):947–963, April 2007.

[8] J. Demmel, Y. Hida, W. Kahan, X.S. Li, S. Mukherjee, and E.J. Riedy. Error bounds from extra-precise iterative refinement. *ACM Trans. Math. Softw.*, 32(2):325–351, June 2006.

[9] George E. Forsythe and Cleve B. Moler. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA, 1967.

[10] M. Gulliksson. Iterative refinement for constrained and weighted linear least squares. *BIT*, 34(2):239–253, 1994.

[11] N. J. Higham. Algorithm 674: FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Trans. Mathematical Software*, 14:381–396, 1988.

[12] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, 1996.

[13] A. Kielbasinski and H. Schwetlick. *Numerische Lineare Algebra: Eine Computerorientierte Einführung*. VEB Deutscher, Berlin, 1988.

[14] Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack Dongarra. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). Technical report, University of Tennessee, Knoxville, TN, June 2006.

[15] 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 Trans. Math. Softw.*, 28(2):152–205, 2002.

[16] W. Oettli and W. Prager. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Num. Math.*, 6:405–409, 1964.

[17] J.K. Reid. Implicit scaling of linear least squares problems. *BIT*, 40(1):146–157, 2000.

[18] J. Rigal and J. Gaches. On the compatibility of a given solution with the data of a linear system. *J. ACM*, 14(3):543–548, 1967.

[19] G. W. Stewart. On the perturbation of pseudo-inverses, projections and linear least squares problems. *SIAM Review*, 19(4):634–662, 1977.

[20] B. Walden, R. Karlson, and Sun. J. Optimal backward perturbation bounds for the linear least squares problem. *Num. Lin. Alg. with Appl.*, 2(3):271–286, 1995.

[21] Roy H. Wampler. A report on the accuracy of some widely used least squares computer programs (in applications). *Journal of the American Statistical Association*, 65(330):549–565, 1970.