Solving $Ax = b$ Dependably with Refinement

(beyond backslash)

Jason Riedy[1]

EECS Department
University of California, Berkeley

28 April, 2008

[1]with Prof. Demmel, Yozo Hida, Xiaoye Li, Meghanath Vishvanath, Deaglan Halligan, Prof. Kahan, and many others.

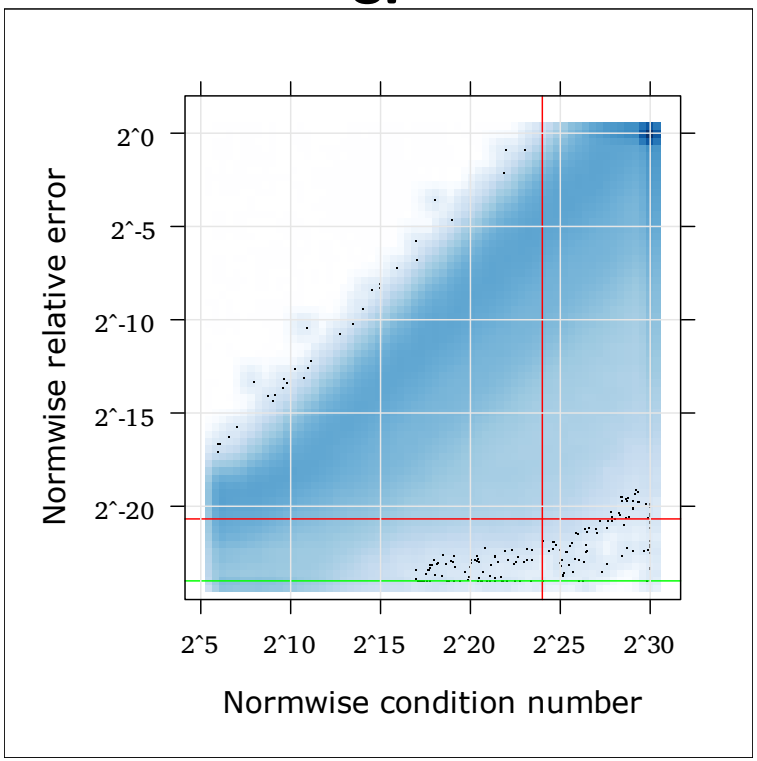
Outline

Who doesn't solve $Ax = b$?

User: `x = A \ b;`

```
warning: matrix singular to machine precision, rcond = 1.69032e-20
warning: attempting to find minimum norm solution
warning: dgelsd: rank deficient 30x30 matrix, rank = 15
```

— or —



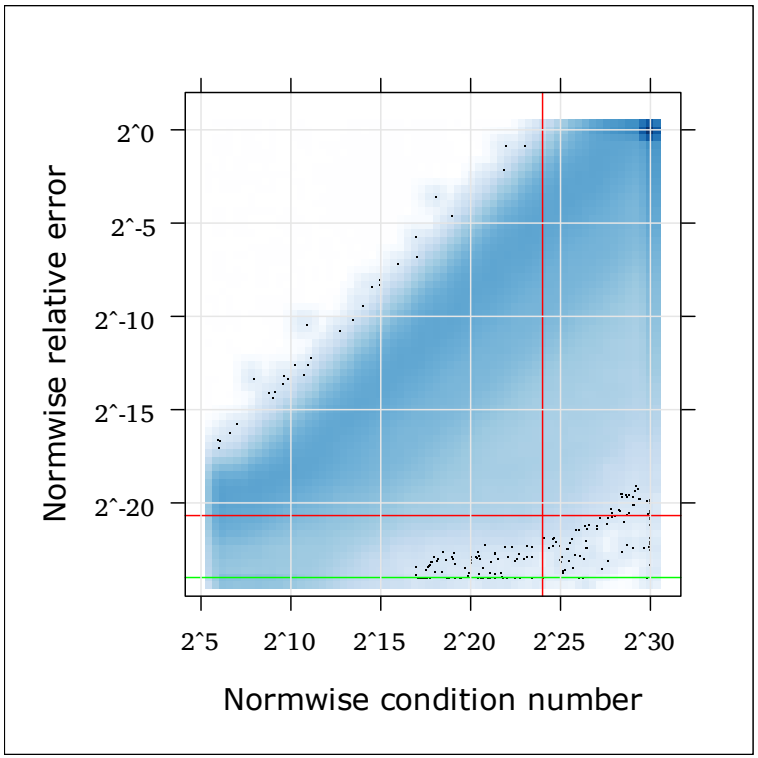
Does that count as success?

Alternatives

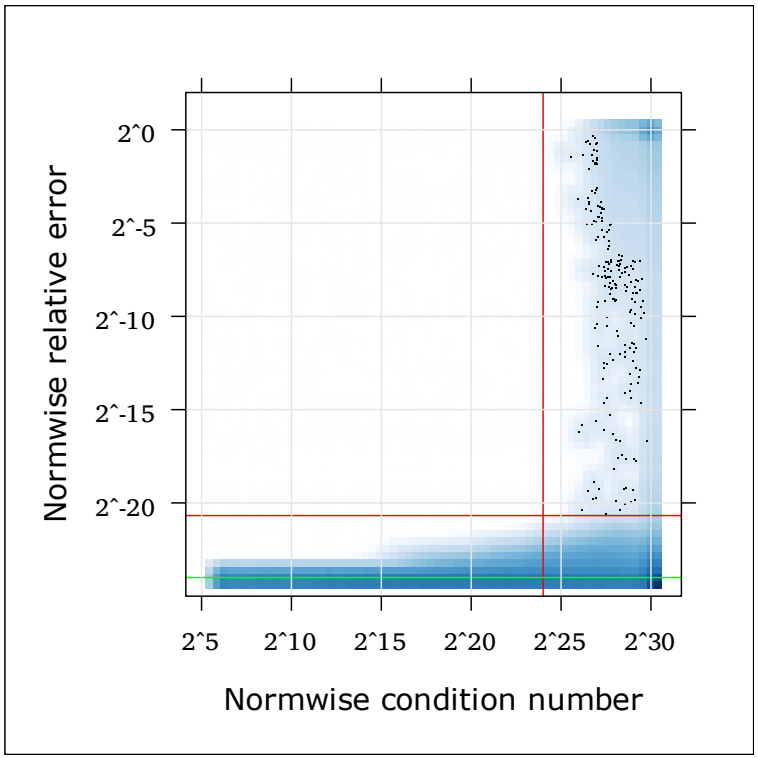
- Interval arithmetic
  - Tells you the answer is wrong, but doesn't correct it.
- Exact arithmetic
  - Exponential in time *and* space: free flops don't matter.
- High-precision arithmetic
  - General-purpose solution
  - Adds to the $O(n^3)$ cost for $Ax = b \ldots$
- Iterative refinement
  - Special-purpose algorithmic change
  - Adds only $O(n^2)$ cost for $Ax = b$.

Iterative refinement works.

Before:



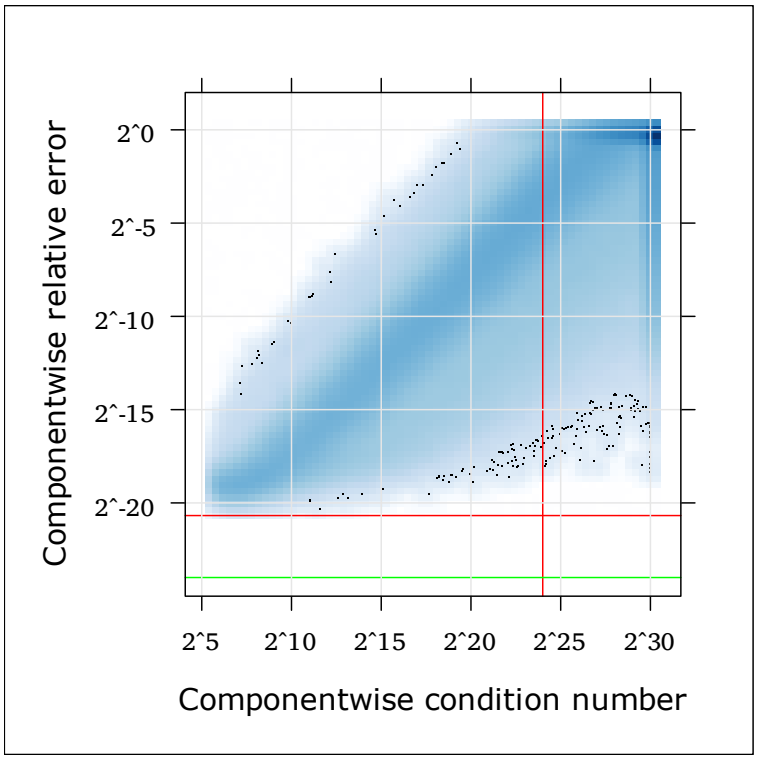
After:



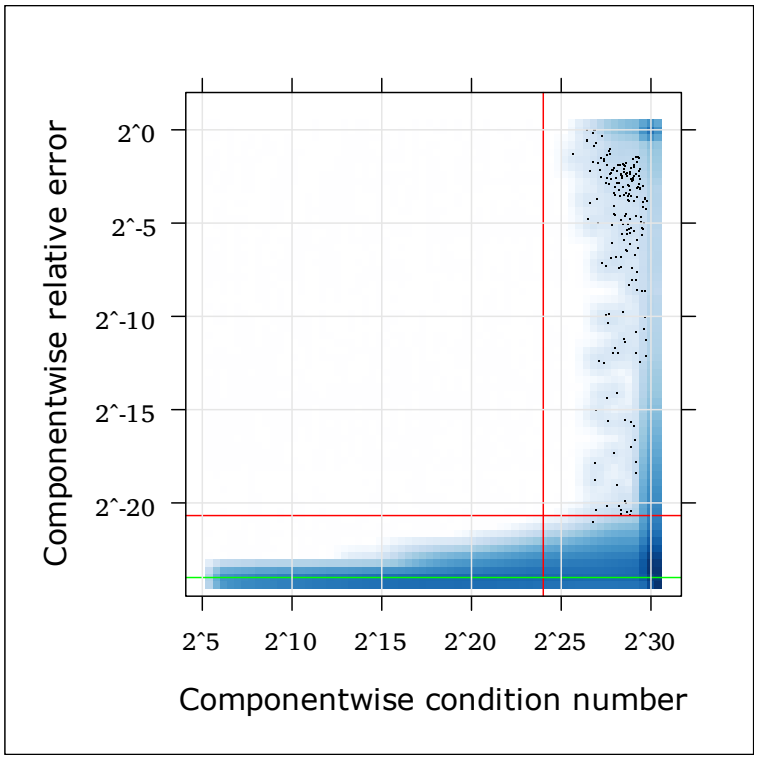
- *Free*: Adds $O(n^2)$ work to $O(n^3)$ algorithm.
- For normwise and *componentwise* error.

Iterative refinement works.

Before:



After:



- *Free*: Adds $O(n^2)$ work to $O(n^3)$ algorithm.
- For normwise and *componentwise* error.

Our changes.

- **Not** a new algorithm...
- Starting with an approximate x:
  1. Compute $r = b - Ax$ with doubled intermediate precision
  2. Solve $Adx = r$ in working precision
  3. Check dx for convergence or no progress
  4. Add $x = x + dx$ to double working precision
- New, extreme details in LAWNs 165 and 188:
  - Targetting *forward* error in x
  - Implicit scaling for *componentwise* results.
  - Carrying x to extra precision.

Roughly speaking...

Error terms: δA, δx, δr in:

$$
\begin{aligned}
\|dx^{(k+1)}\| = \|\overbrace{\left(A+\delta A^{(k+1)}\right)^{-1}\delta A^{(k)}}^{\text{convergence rate}} dx^{(k)} \\
+\left(A+\delta A^{(k+1)}\right)^{-1}\delta A^{(k)}\delta x^{(k)} \\
+\left(A+\delta A^{(k+1)}\right)^{-1}\delta r^{(k+1)}\| \quad \Big\} \text{ force tiny} \\
\approx \rho^{(k)}\left(\|dx^{(k)}\|+O(\varepsilon^2)\right) \\
\lessapprox \varepsilon c_n g\,\mathrm{cond}\#(A)\left(\|dx^{(k)}\|+O(\varepsilon^2)\right)
\end{aligned}
$$

When $dx^{(k+1)}$ is small relative to $x^{(k)}$, converged.
If $\rho^{(k)}$ grows too large, ill-conditioned or other errors encroaching.

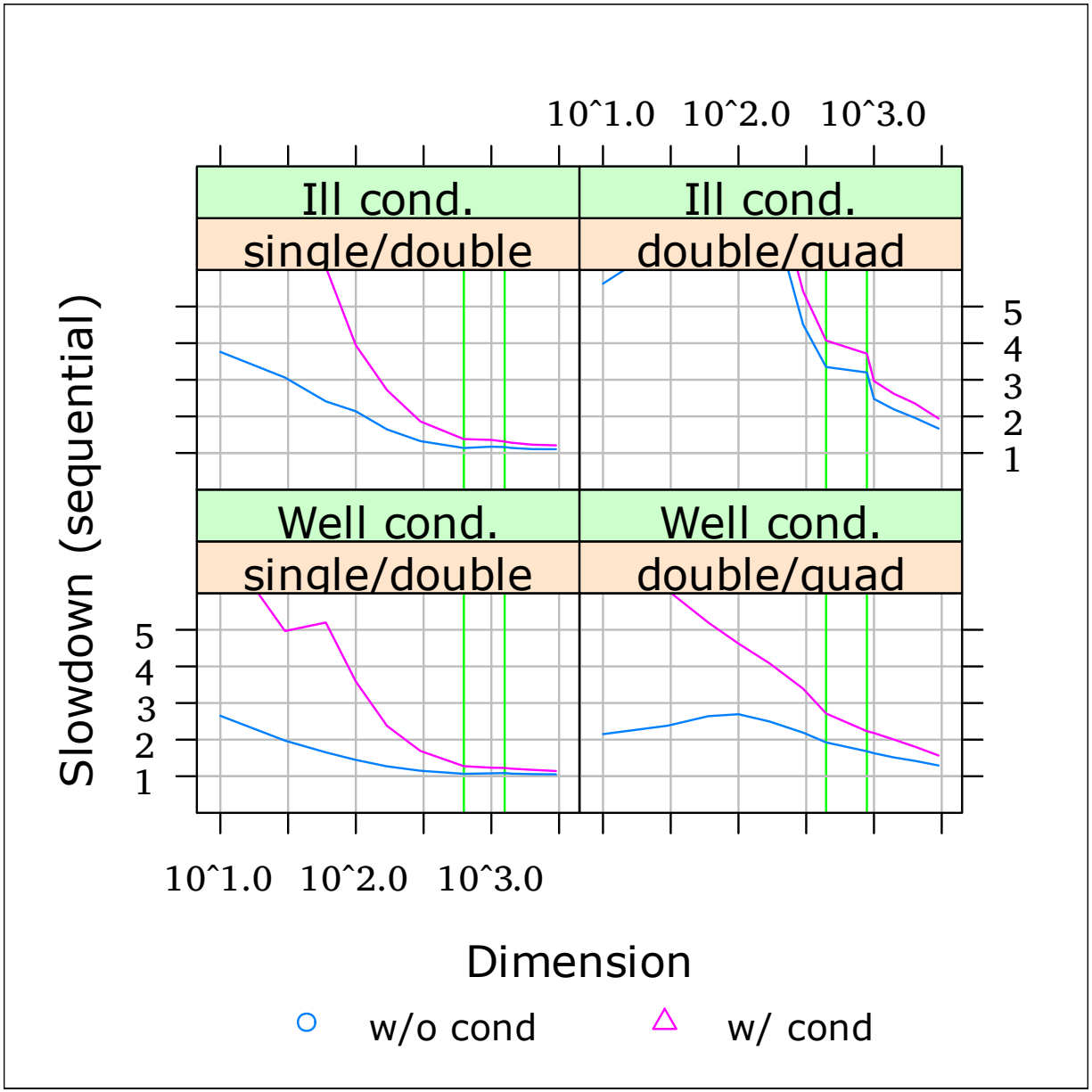
Other little details

- Componentwise results by implicit scaling:
  - Solve $(A\,\mathrm{diag}(x))z = b$, where $z = 1$...
  - Approximate scaling with current x: step $dz \approx \mathrm{diag}(x^{(k)})^{-1} dx$.
  - Treat componentwise results as a scaled norm.
  - (Could work for any scaled norm, but interface nightmare.)
- Equilibration:
  - Scale A by row and column factors, $A_s = D_r A D_c$; $b_s = D_r b$.
  - Scale step size back by D_c.
  - Reduces incidence of *artificial* ill-conditioning.
  - **Very** important: symmetric indefinite; sparse

It slices, it dices...

- Applies to overdetermined least squares problems.
  - Verified for typical $\min \|b - Ax\|_2$.
  - Should work for weighted and constrained.
- More freedom for sparse ordering, etc.
  - Use refinement to "clean up" a restricted factorization.
  - Can perturb to avoid entry growth.
- Can help other low-precision factorizations.
  - UTK: Single-prec factorization gives double-prec residual.
  - **Note:** Assume column-scaling invariance, no Strassen...

Refinement: cheap. Dependability: costly.



Impact on parallelism / communication

- Refinement:
  - Residual calculations: If x is replicated, no communication.
  - Solving $Adx = r$: Use communication-avoiding factorization, etc.
  - Updating x: $\log P$ term for scattering dx pieces.

Overall: Still negligible compared to factorization.

- Norm estimation:
  - Two solves (A and A^*) per iteration
  - Multiple global reductions per iteration
  - Often as many iterations as refinement steps

Swarm of negligible operations becomes expensive.

Lowering costs of dependability

- Use the convergence rate:
  - Convergence rate $\lessapprox \varepsilon\, c_n\, g\ \mathrm{cond}\#(A)$
  - Failure to converge $\Rightarrow$ ill-conditioned / lousy solver
  - Need a few steps to *find* the convergence rate.
    - ⋆ Singular A: many vectors x may "converge" immediately.
  - Fails to detect failure in 0.006% of our excruciating tests.
  - Each failure: ill-conditioned, converged immediately!
- Use a second vector $y^{(1)} \neq x^{(1)}$:
  - Only when x converged immediately...
  - Should converge to same final solution.
    - ⋆ Obtaining two different solutions $\Rightarrow$ singular A
  - Only requires flops and bandwidth...
  - (Work in progress.)
- Both can be applied without solving with A^*.

Feedback needed!

- **What data is provided to users?**
  - Solution and failure flag, obviously.
  - Could estimate condition numbers, growth factors, ...
- **What data is expected from users?**
  - Should numerical scaling / equilibration be automatic?
- **What interface?**
  - Remember Prof. Demmel's worst-case? It's this code.
  - High-level interface is frozen once in Sca/LAPACK.
  - Must be generally applicable, easy to support.

Relevant locations

Iterative refinement test drivers with excruciating testing framework:

- Papers:
  `http://www.netlib.org/lapack/lawnspdf/lawn165.pdf`
  `http://www.netlib.org/lapack/lawnspdf/lawn188.pdf`
- Development code:
  `http://www.cs.berkeley.edu/~ejr/dev/lapack/itref-work/`
- Git repositories:
  `http://www.cs.berkeley.edu/~ejr/gits/itref-work.git/`
  `http://www.cs.berkeley.edu/~ejr/gits/itref-lsq.git/`

XBLAS (extended-precision BLAS):

- Release code: `http://crd.lbl.gov/~xiaoye/XBLAS/`
- Git repository:
  `http://www.cs.berkeley.edu/~yozo/gits/xblas.git/`