# Dependable direct solutions for linear systems using a little extra precision

E. Jason Riedy
jason.riedy@cc.gatech.edu

CSE Division, College of Computing
Georgia Institute of Technology

21 August, 2009

# $Ax = b$

## Workhorse in scientific computing

- Two primary linear algebra problems: $Ax = b$, $Av = \lambda v$
- Many applications reduce problems into those, often $Ax = b$.
  - PDEs: Discretize to one of the above.
  - Optimization: Solve one at each step.
  - ...
- Commercial supercomputers are *built* for $Ax = b$: LINPACK

Many people work to solve $Ax = b$ *faster*.
Today's focus is solving it **better**.

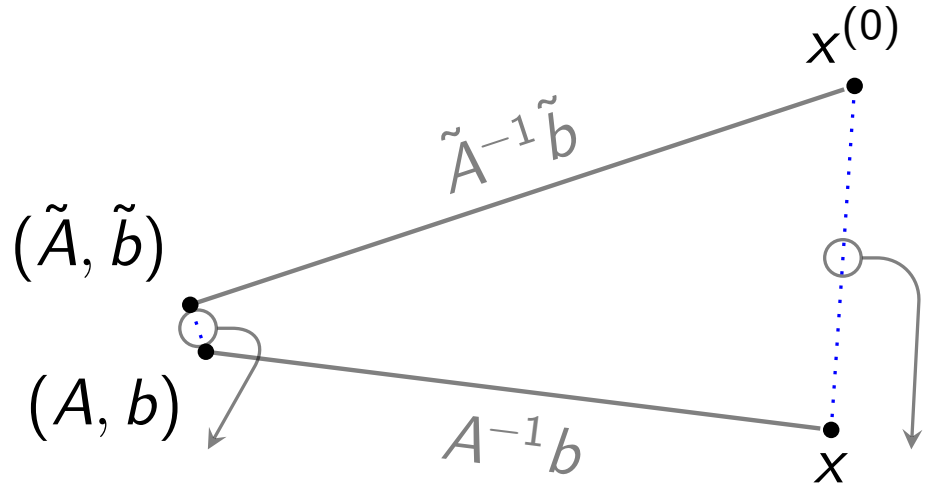
(I'm oversimplifying in many ways. And better can lead to faster.)

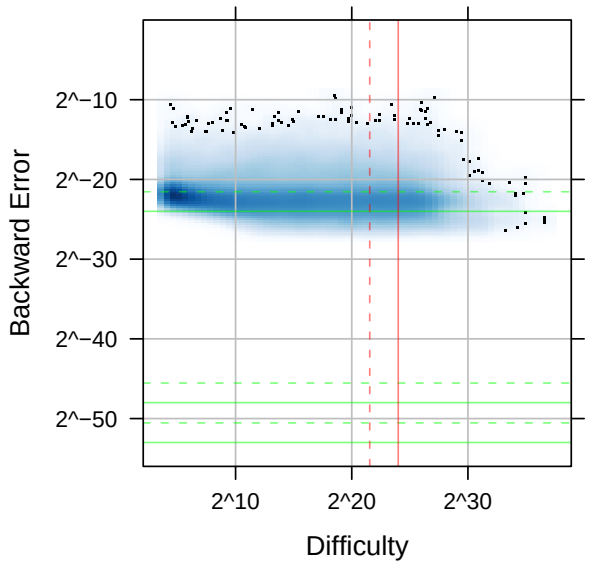
# Outline

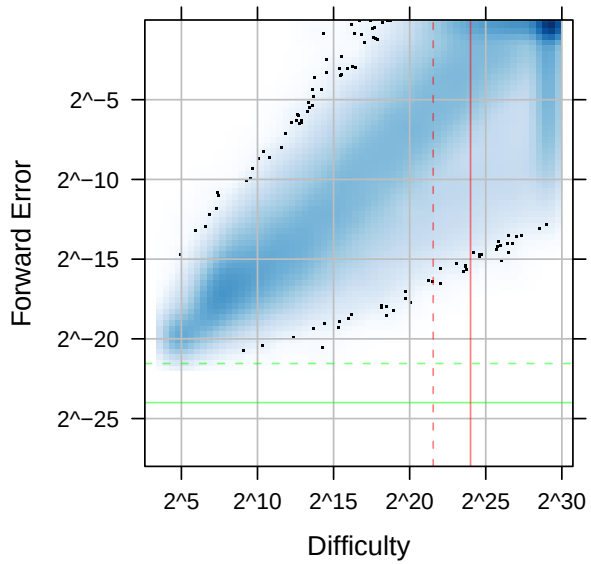
1. What do I mean by "better"?
2. Refining to more accurate solutions with extra precision
3. Other applications of better: faster, more scalable

Most of this work was done at/with UC Berkeley in conjunction with Yozo Hida, Dr. James Demmel, Dr. Xiaoye Li (LBL), and a long sequence of then-undergrads (M. Vishvanath, D. Lu, D. Halligan, ...).

# Errors in $Ax = b$



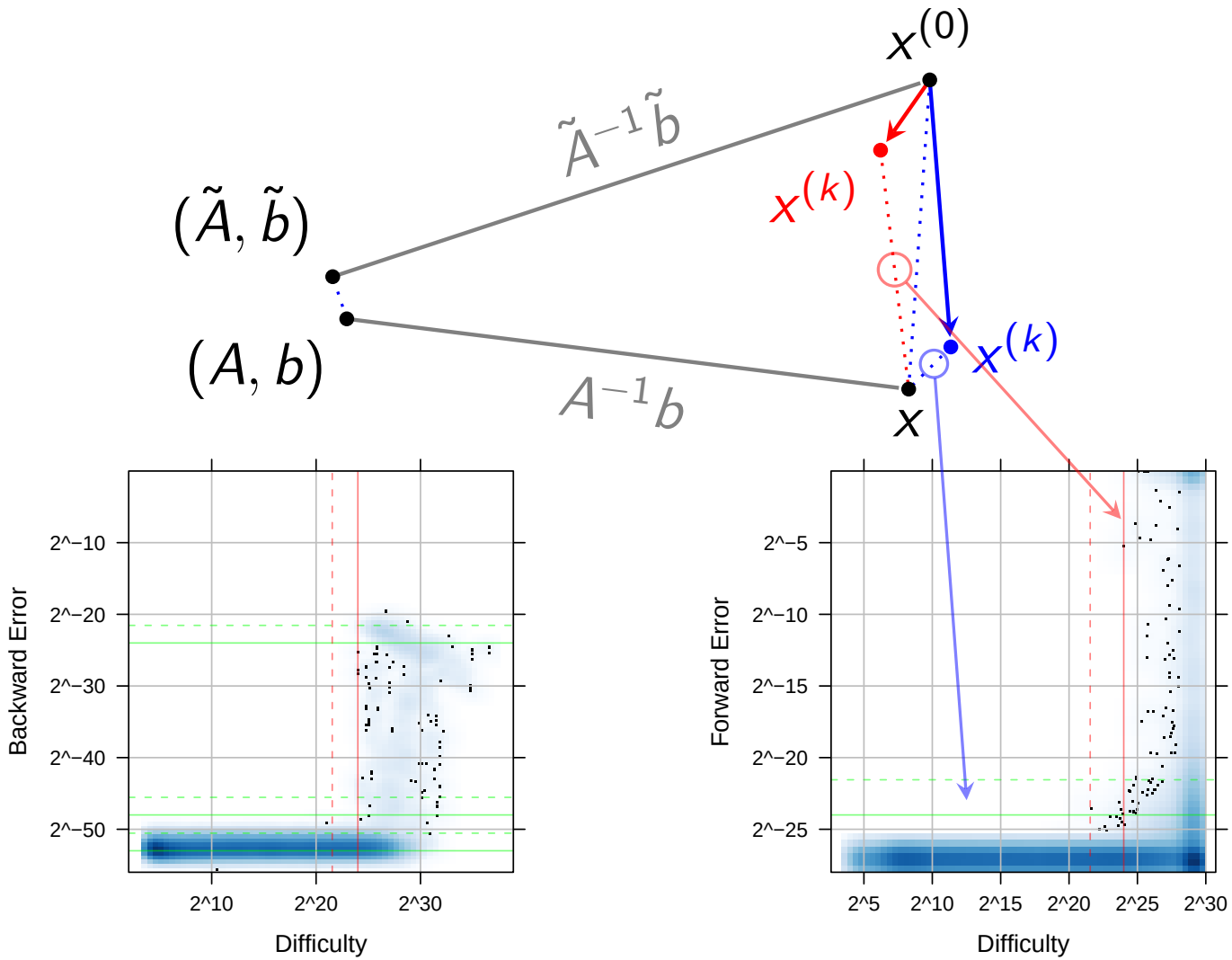




backward error (berr)

forward error (ferr)

# Goals for errors in $Ax = b$




backward error (berr)

forward error (ferr)

# Possible methods

- Interval arithmetic
  - Tells you when there's a problem, not how to solve it.
  - Finding the optimal enclosure is NP-hard!
- Exact / rational arithmetic
  - Storage (& computation) grows exponentially with dimension.
- Telescoping precisions (increase precision throughout)
  - Increases the cost of the $O(n^3)$ portion.
- **Iterative refinement**
  - $O(n^2)$ extra work after $O(n^3)$ factorization.
  - Only a little extra precision necessary!
  - Downside: Dependable, but not validated.

## Dependable solver

Reduce the error to the precision's limit as often as reasonable, or clearly indicate when the result is unsure.

# What I'm **not** going to explain deeply

- Precise definition of *difficulty*:
  - A condition number relevant to the error in consideration, or,
  - roughly, the error measure's sensitivity to perturbation near the solution.
- Numerical scaling / equilibration:
  - Assume all numbers in the input are roughly in the same scale.
  - Rarely true for computer-produced problems.
  - Common cases easy to handle; obscures the important points.
  - *Note:* Poor scaling produces *simple* ill-conditioning.
- Details of when each error measure is appropriate.
  - Backward: normwise, columnwise, **componentwise**, ...
  - Forward: normwise, **componentwise**, ...

**All three are inter-linked and address norms.**

# Outline

# Iterative refinement

## Newton's method for $Ax = b$

Assume $A$ is $n \times n$, non-singular, factored $PA = LU$, *etc.*

1. Solve $Ax^{(0)} = b$
2. Repeat for $i = 0, \ldots$:
   1. Compute residual $r^{(i)} = b - Ax^{(i)}$.
   2. (Check backward error criteria)
   3. Solve $Adx^{(i)} = r^{(i)}$.
   4. (Check forward error criteria)
   5. Update $x^{(i+1)} = x^{(i)} + dx^{(i)}$.

- Overall algorithm is well-known (Forsythe & Moler, 1967...).
- In exact arithmetic, would converge in one step.

# Iterative refinement

## Newton's method for $Ax = b$

Assume $A$ is $n \times n$, non-singular, factored $PA = LU$, *etc.*

1. Solve $Ax^{(0)} = b$
2. Repeat for $i = 0, \ldots$:
   1. Compute residual $r^{(i)} = b - Ax^{(i)}$. **(Using double precision.)**
   2. (Check backward error criteria)
   3. Solve $Adx^{(i)} = r^{(i)}$. **(Using working/single precision.)**
   4. (Check forward error criteria)
   5. Update $x^{(i+1)} = x^{(i)} + dx^{(i)}$. **(New: x with double precision.)**

- No extra precision: Reduce backward error in one step [Skeel].
- A bit of double precision: Reduce errors much, much further.

# Why should this work?

A *brief*, informal excursion into the analysis...

- $r^{(i)} = b - Ax^{(i)} + \delta r^{(i)}$
- $(A + \delta A^{(i)})dx^{(i)} = r^{(i)}$
- $x^{(i+1)} = x^{(i)} + dx^{(i)} + \delta x^{(i+1)}$

**Very** roughly (**not** correct, approximating behavior, see LAWN165):

## Backward Error (Residual)

$$r^{(i+1)} \approx \varepsilon_w A\, A^{-1}\, r^{(i)} + A\, \delta x^{(i)} + \delta r^{(i)}$$

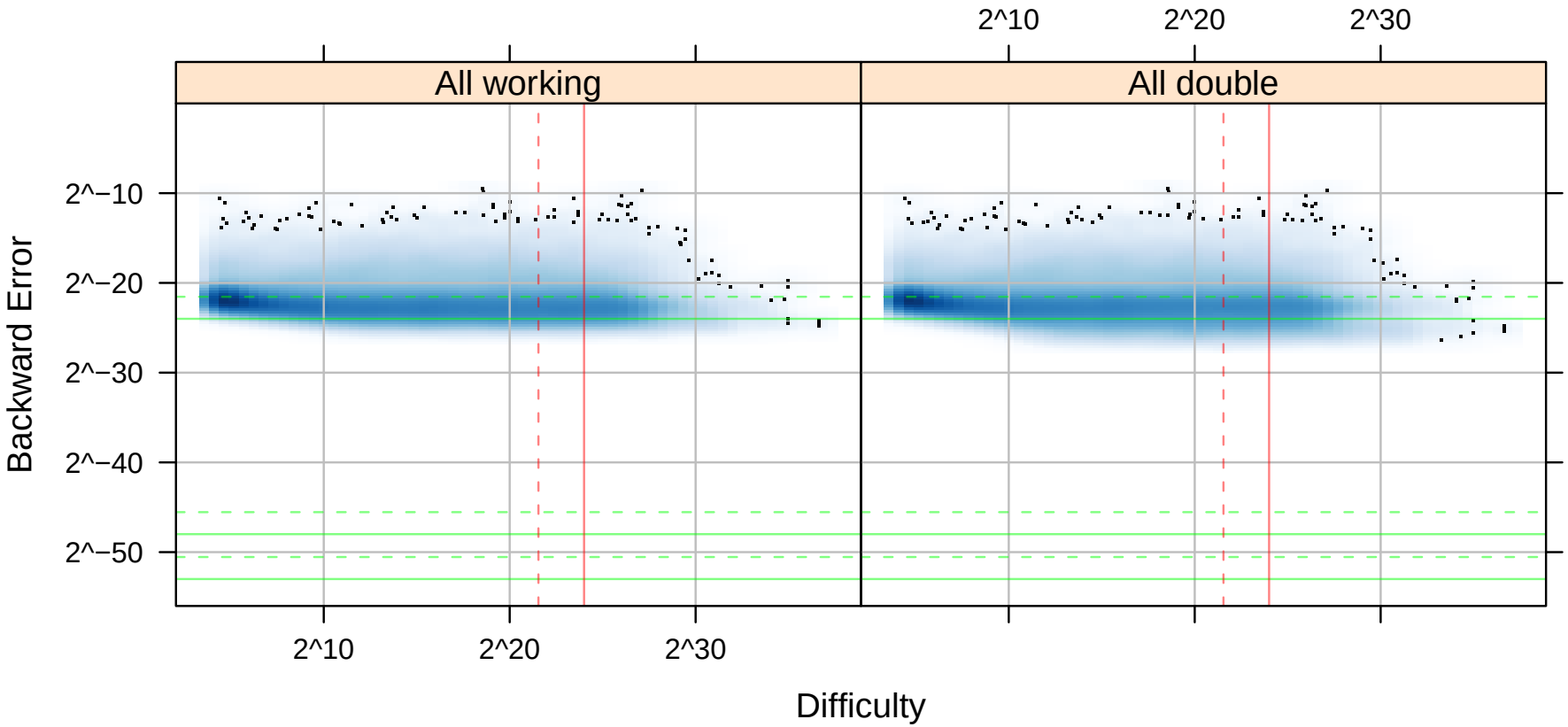
## Forward Error

$$e^{(i+1)} \approx \varepsilon_w A^{-1}\, A\, e^{(i)} + \delta x^{(i)} + A^{-1}\, \delta r^{(i)}$$

# Test cases

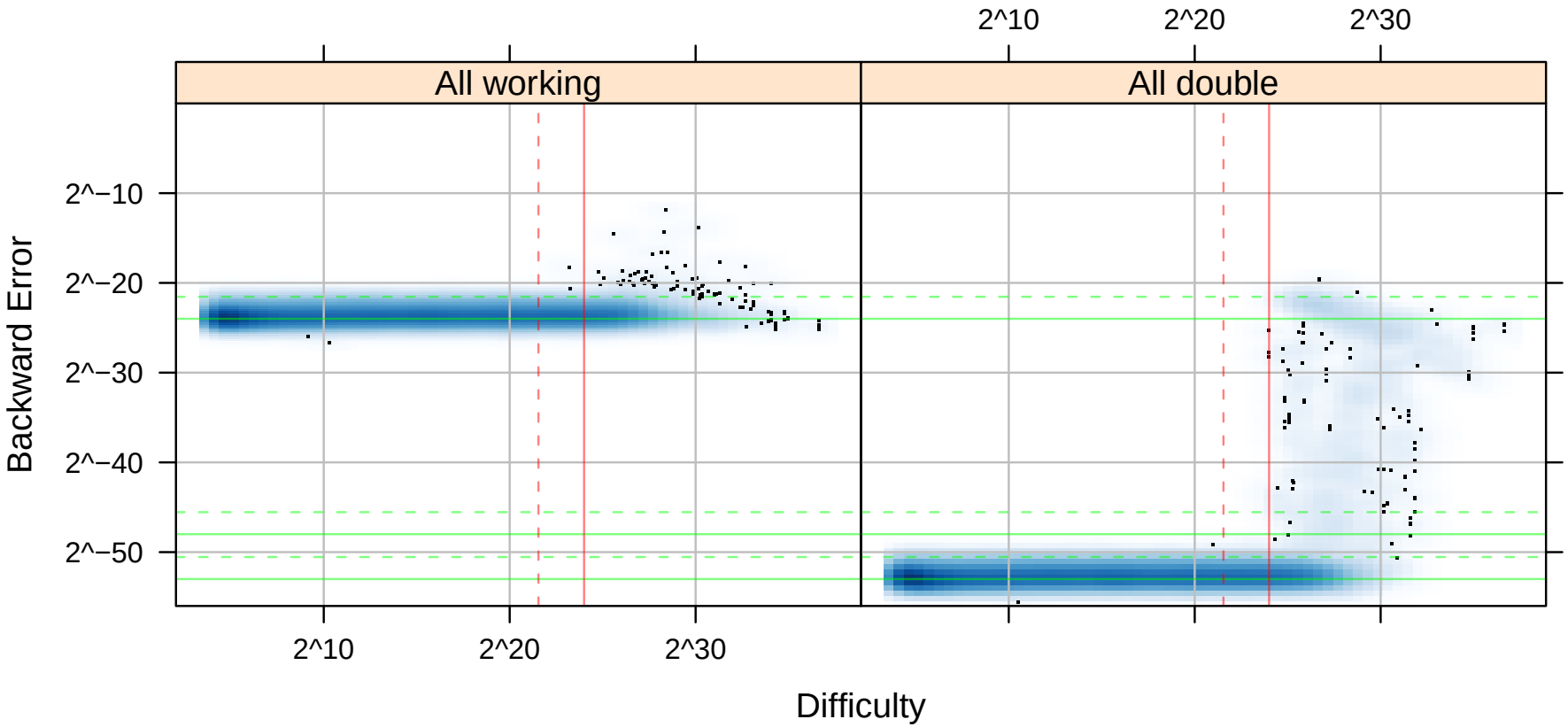
- One million random, single-precision, $30 \times 30$ systems $Ax = b$
  - 250k: $A$ generated to cover factorization difficulty
  - Four $(x, b)$, two with random $x$ and two with random $b$
  - Solve for true $x$ using greater than quad precision.
  - Working precision: $\varepsilon_w = 2^{-24} \approx 6 \times 10^{-8}$
  - Extra / double precision: $\varepsilon_x = 2^{-53} \approx 10^{-16}$
- Using single precision and small because
  - generating and running one million tests takes time, and also
  - it's easier to hit difficult cases!
- Results apply to double, complex & double complex (with $2\sqrt{2}$ factor). Also on tests (fewer) running beyond $1k \times 1k$.
- Note: Same plots apply to sparse matrices in various collections, but far fewer than 1M test cases.

# Backward error results (before)



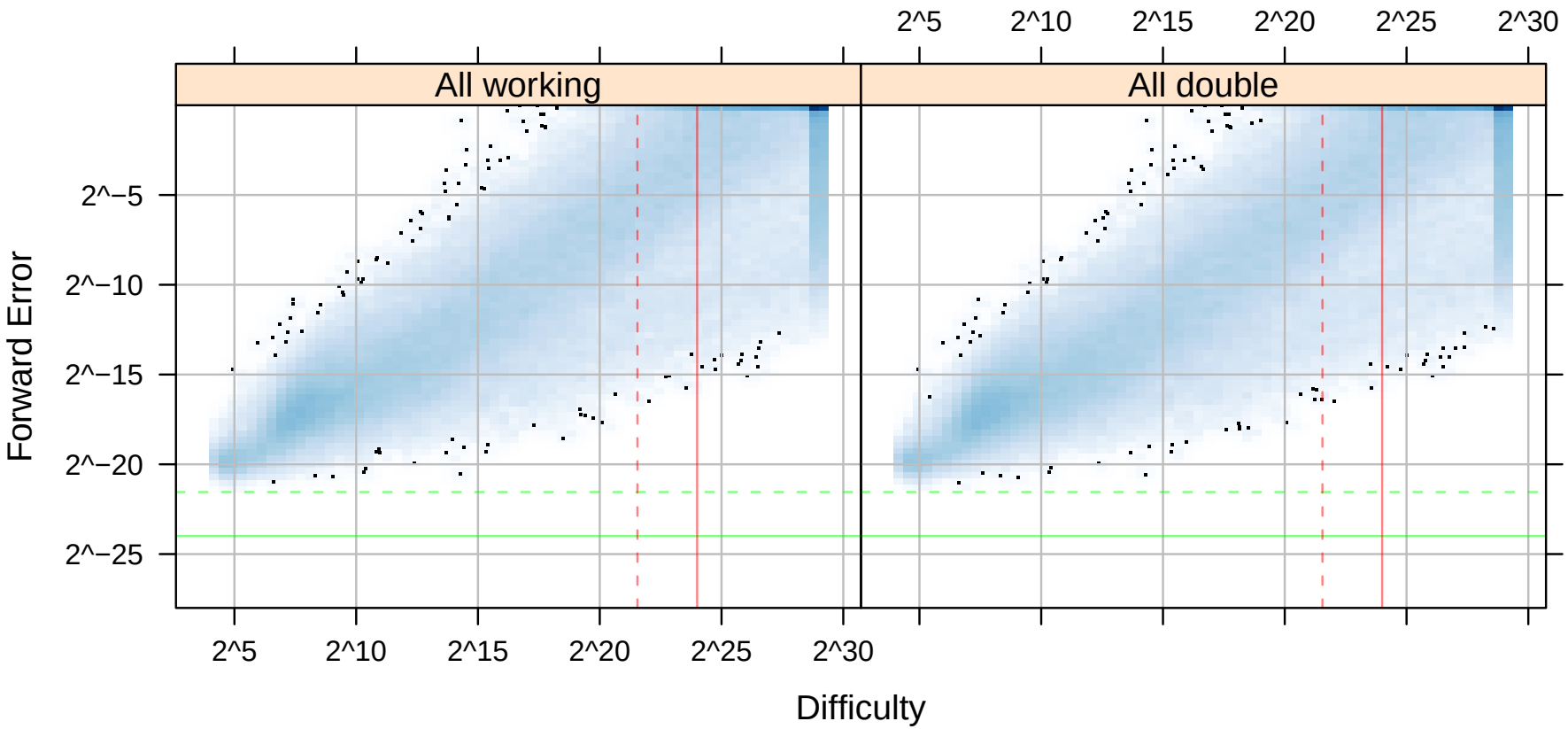

- Omitting double-prec residuals; same limiting error as all working.
- All-double backward error is for the double-prec $x$.

# Backward error results (after)



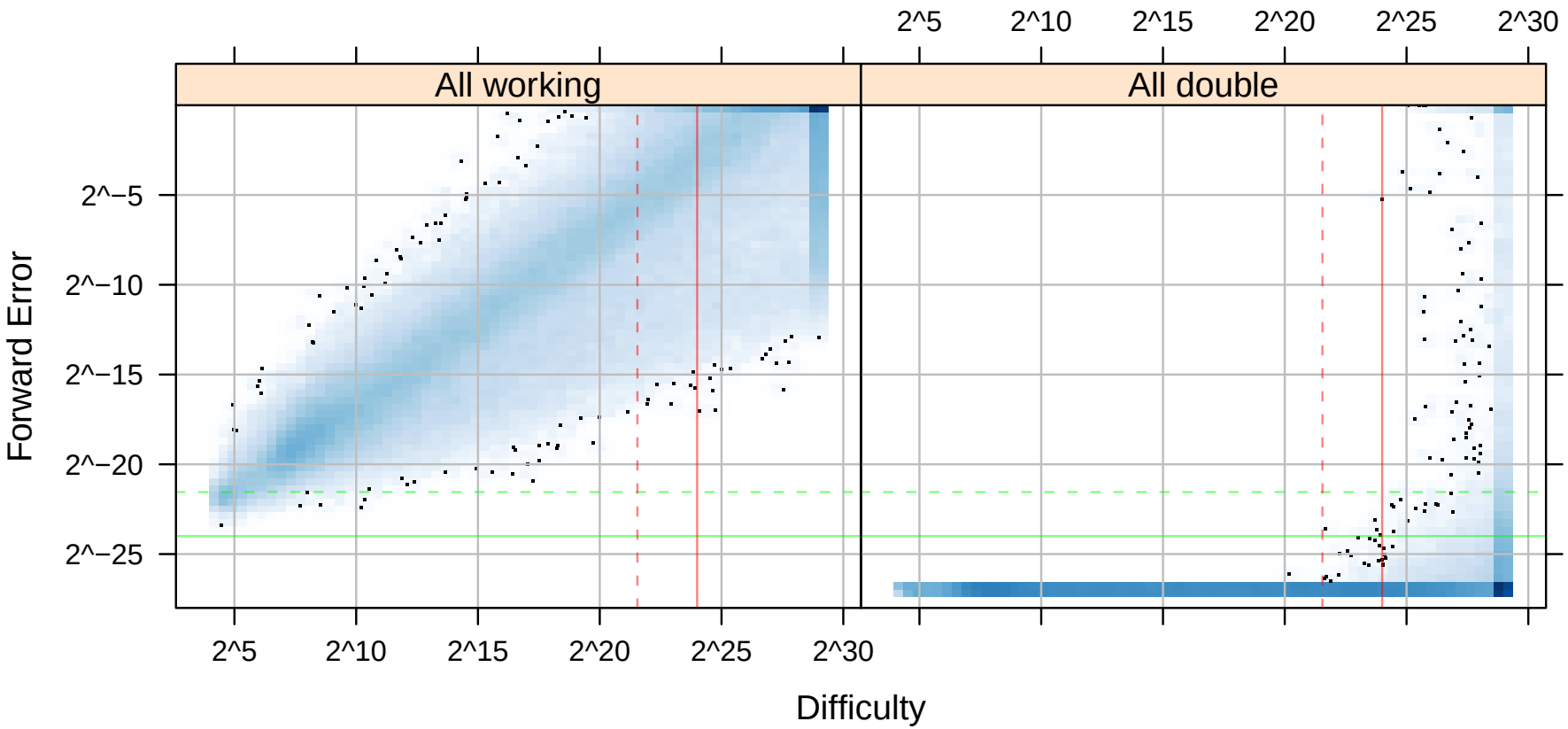

- Omitting double-prec residuals; same limiting error as all working.
- All-double backward error is for the double-prec $x$.

# Forward error results (before)




- Omitting double-prec residuals; same limiting error as all working.
- All-double *forward* error is for the *single*-prec $x$.

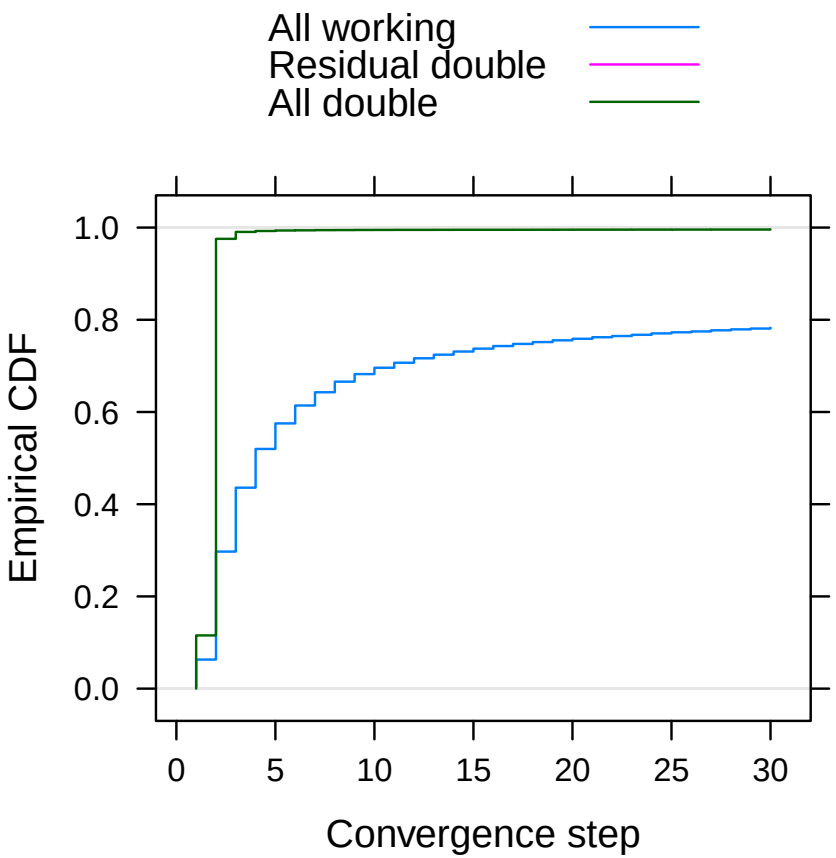
# Forward error results (after)



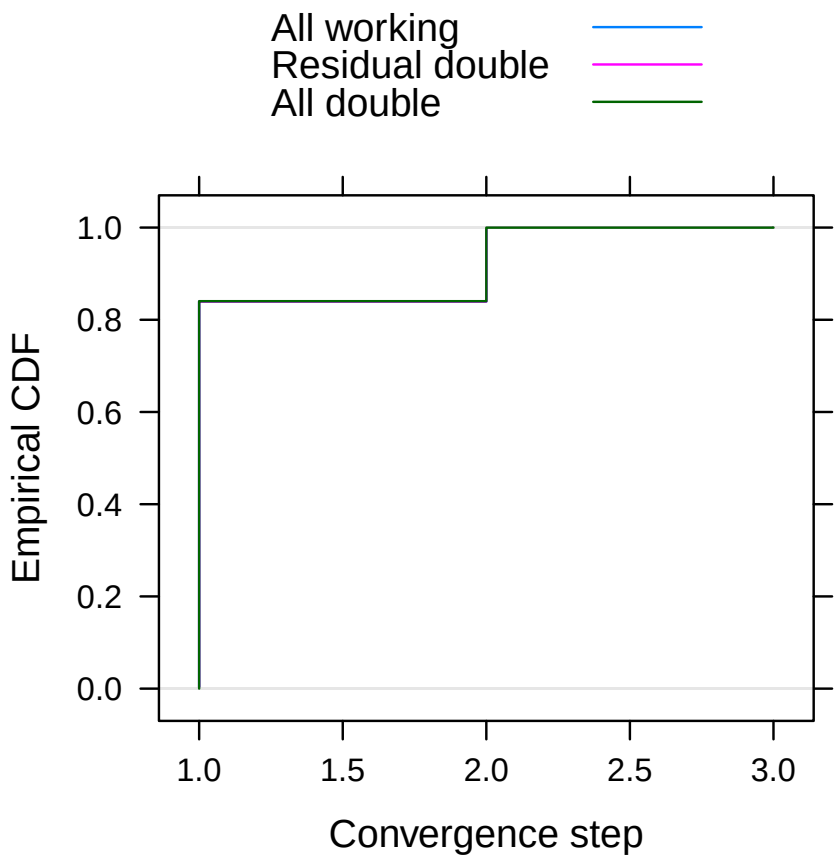

- Omitting double-prec residuals; same limiting error as all working.
- All-double *forward* error is for the *single*-prec $x$.

# Iteration costs: backward error

## Convergence to $\varepsilon_{\mathbf{w}}$



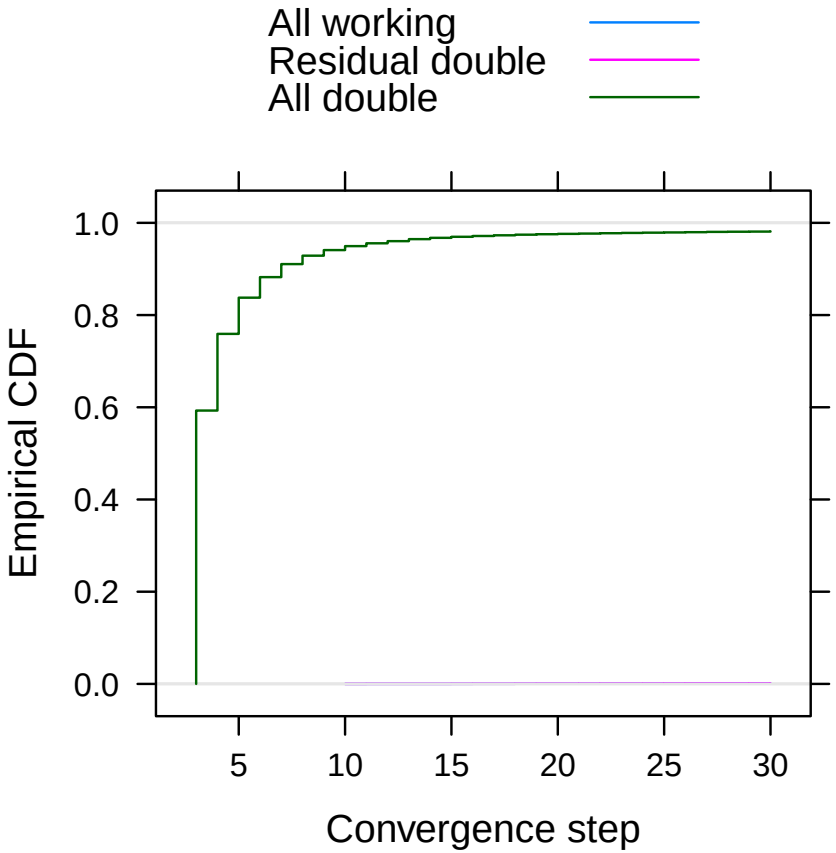

## Convergence to $10\varepsilon_{\mathbf{w}}$



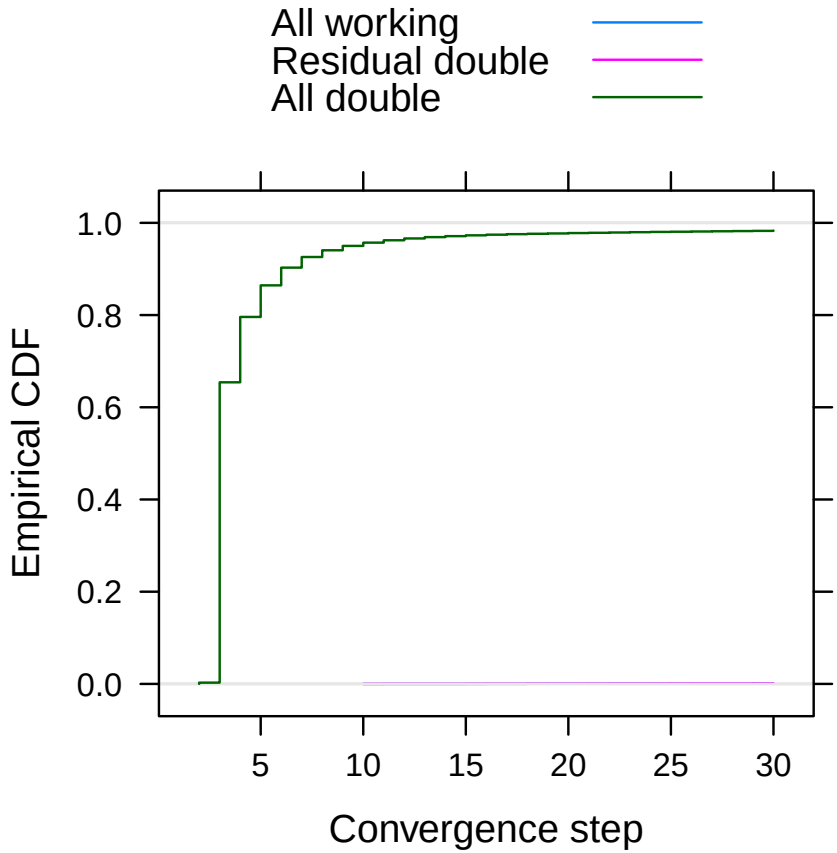

Practical: Stop when backward error is tiny or makes little progress.

# Iteration costs: backward error

**Convergence to $\varepsilon_{\mathbf{w}}^2$**



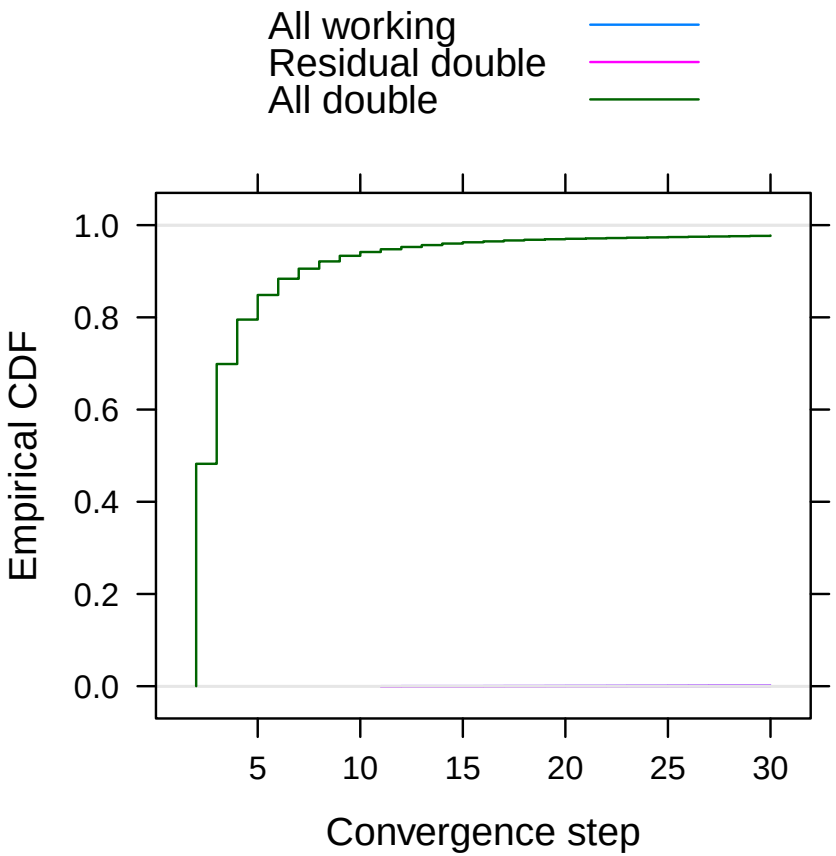

**Convergence to $10\varepsilon_{\mathbf{w}}^2$**



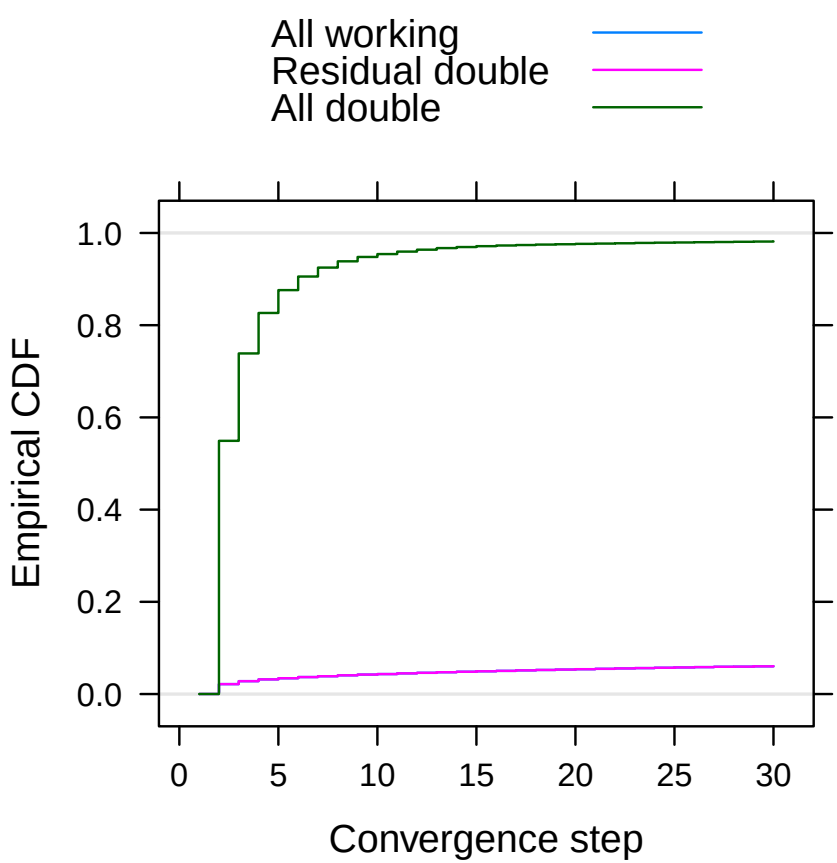

Practical: Stop when backward error is tiny or makes little progress.

# Iteration costs: forward error

**Convergence to $\varepsilon_{\mathbf{w}}$**



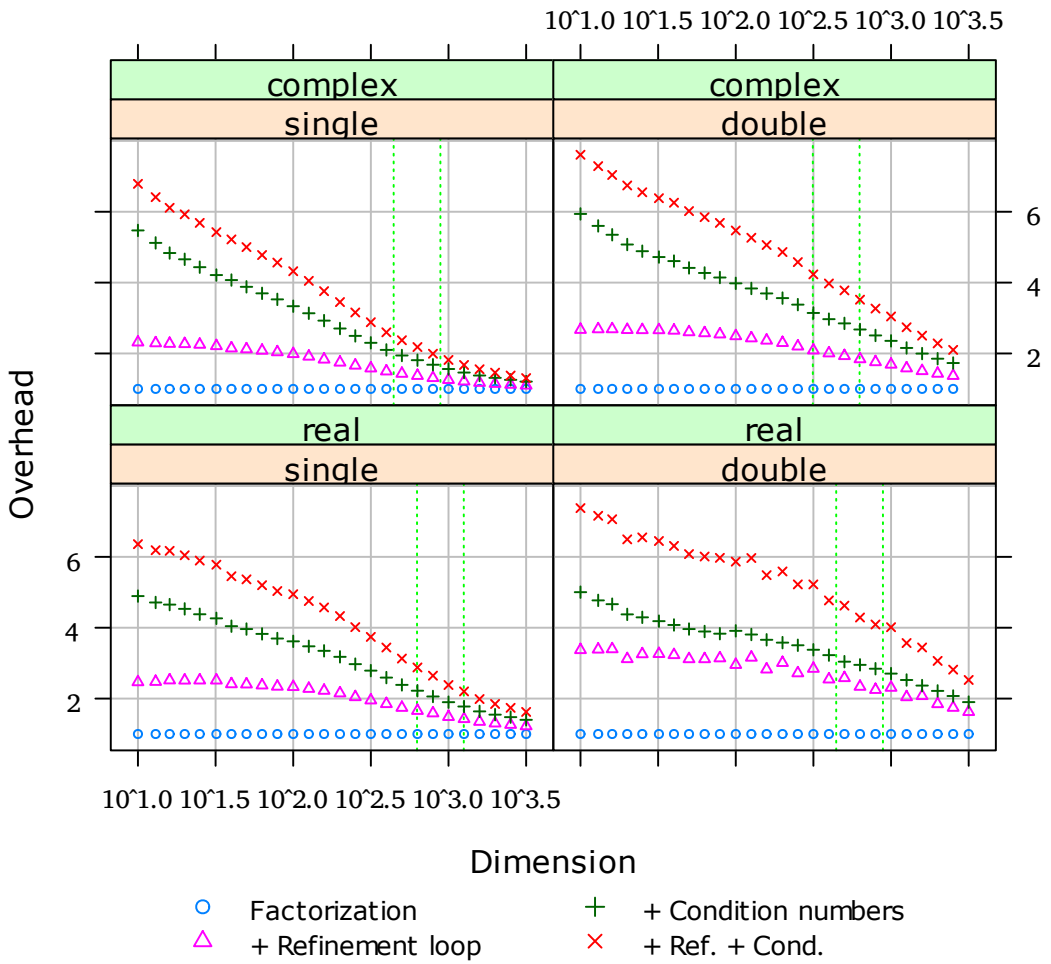

**Convergence to $\sqrt{\mathbf{N}} \cdot \varepsilon_{\mathbf{w}}$**




Practical: Stop when $dx$ is tiny or makes little progress.

# Performance costs

**Overhead each phase by precision and type**



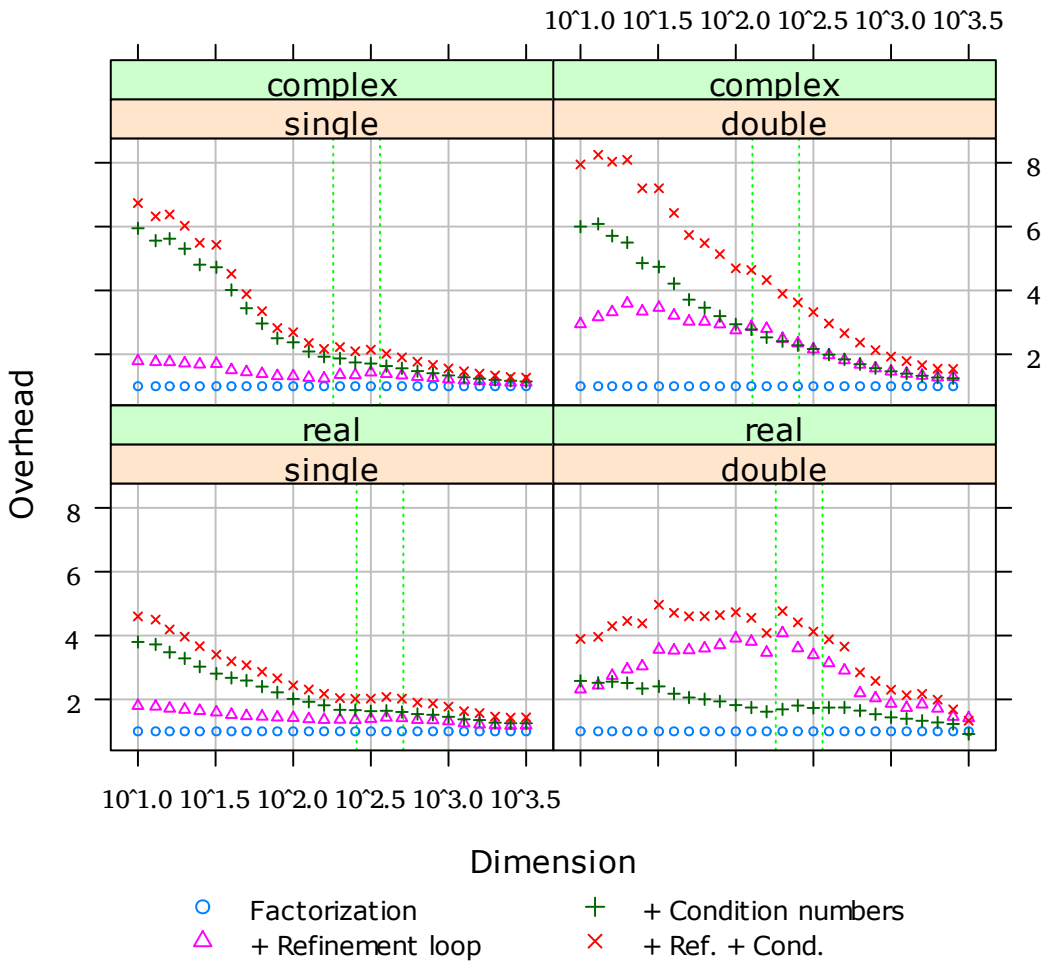

**Overhead is time for phase (incl. fact.) / time for factorization**

## Itanium 2

- Relatively balanced cpu / mem arch.
- Double *faster* than single

# Performance costs

**Overhead each phase by precision and type**




**Overhead is time for phase (incl. fact.) / time for factorization**

## Xeon 3GHz

- Horribly unbalanced cpu / mem arch.
- (Not parallel)
- Vector instructions
- No vectorization in extra precision ops.

# Outline

# Obvious applications of better

## Available in LAPACK

- Routines `SGESVXX`, `DGESVXX`, `CGESVXX`, `ZGESVXX`
- *Experimental* interface, subject to changes

## High-level environments

- Do you want to think about all error conditions all the time?
- Should be in Octave & MATLAB™:

```
x = A\b;
```

- The same technique applies to overdetermined least-squares [LAWN188; Demmel, Hida, Li, Riedy]. R or S+ (statistics):

```
model <- lm(response~var)
```

  - Refine the augmented system $\begin{bmatrix} A & \alpha I \\ 0 & A^T \end{bmatrix} \begin{bmatrix} x \\ r/\alpha \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$. [Björck]

# Not so obvious application: Speed!

## When single precision is much faster than double...

- Assume: Targeting backward error, often well-conditioned
- Factor $A$ in single precision, use for $Adx_i = r$.
- Refine to dp backward error, or fall back to using dp overall.
- Earlier Cell (extra slow double): 12 Gflop/s $\Rightarrow$ 150 Gflop/s! [LAWN175; Langou[2], Luszczek, Kurzak, Buttari, Dongarra]
- (Independent path to the same destination.)

## When single precision fits more into memory...

- Sparse, sparse out-of-core
  - Generally limited by indexing performance [Hogg & Scott]
  - Could use packed data structures from Cell [Williams, *et al.*]

# Not so obvious application: Scalability!

## When pivoting is a major bottleneck...

- Sparse, unsymmetric $LU$ factorization:
  - Completely separate structural analysis from numerical work.
  - Introduce backward errors to avoid *entry growth*.
  - Fix with refinement.
  - (SuperLU [Demmel, Li, (+ me)], earlier sym.indef. work)

## When pivoting blocks *practical* theory...

- Communication-optimal algorithms for $O(n^3)$ linear algebra
  - Trade some computation for optimal memory transfers / comm. [LAWN218; Ballard, Demmel, Holtz, Schwartz]
  - Codes exist, are fast, *etc.*
- But $LU$ cannot use partial pivoting!
  - Use a new strategy [Demmel, Grigori, Xiang], refine...

# Summary

- We can construct an inexpensive, *dependable* solver for $Ax = b$.
  - Compute an *accurate* answer whenever feasible.
  - Reliably detect failures / unsure, even for the forward error.
- We can compute *better* results for $Ax = b$.
  - Trade some computation, a little bandwidth for accuracy.
  - Important bit is keeping *all* the limiting terms (residual, solution) to extra precision
- Better results can help solve $Ax = b$ more quickly.
  - Start with a sloppy solver and fix it.

# Questions / Backup

# Doubled-precision

- Represent $a \circ b$ exactly as a pair $(h, t)$.
- Old algorithms [Knuth, Dekker, Linnainmaa, Kahan; 60s & 70s]
- Work on any *faithful* arithmetic [Priest]

## Addition

- $h = a + b$
- $z = h - a$
- $t = (a - (h - z)) + (b - z)$

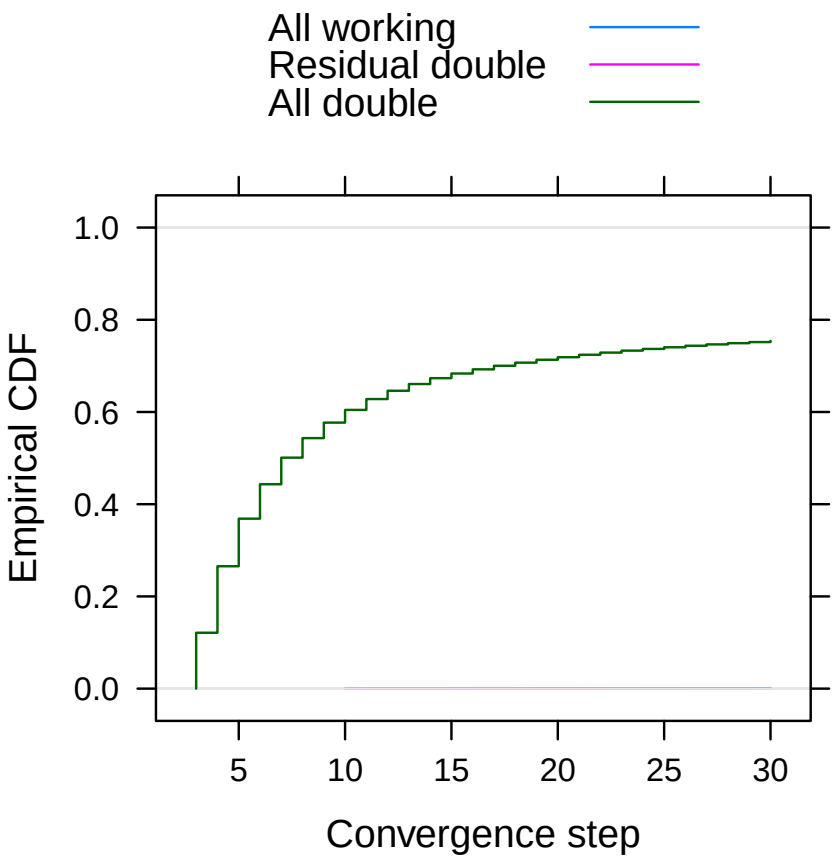
## Multiplication

- $h = a \cdot b$
- $(ah, at) = split(a)$
- $(bh, bt) = split(b)$
- $t = ah \cdot at - h$
- $t = ((t + (ah * bt)) + (at * bh)) + (at * bt)$

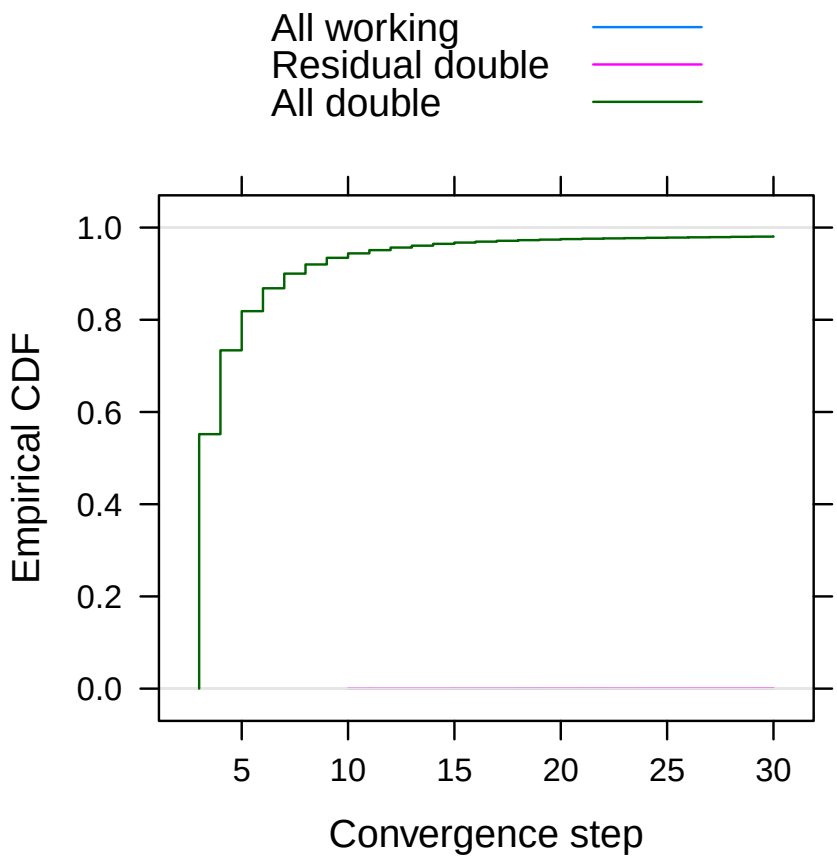
See qd package from [Bailey, Hida, Li]; recent pubs from [Rump, Ogita, Oishi].

# Iteration costs: backward error to double

**Convergence to $\varepsilon_{\mathbf{x}}$**




**Convergence to $10\varepsilon_{\mathbf{x}}$**




Practical: Stop when backward error is tiny or makes little progress.