MANCHESTER 1824

The University of Manchester

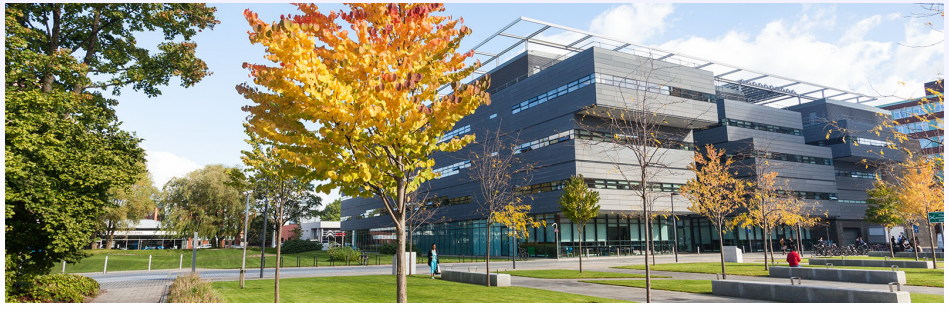
M\cr NA
Manchester Numerical Analysis

Multiprecision Algorithms

Nick Higham
School of Mathematics
The University of Manchester

`http://www.maths.manchester.ac.uk/~higham/`
`@nhigham, nickhigham.wordpress.com`

Mathematical Modelling, Numerical Analysis and Scientific Computing, Kácov, Czech Republic
May 27-June 1, 2018.



Outline

Multiprecision arithmetic: floating point arithmetic supporting multiple, possibly arbitrary, precisions.

- Applications of & support for low precision.
- Applications of & support for high precision.
- How to exploit different precisions to achieve faster algs with higher accuracy.
- Focus on
  - **iterative refinement** for $Ax = b$,
  - **matrix logarithm**.

Download these slides from `http://bit.ly/kacov18`

Lecture 1

- Floating-point arithmetic.
- Hardware landscape.
- Low precision arithmetic.

siam news

MORE AT SIAM

SIAM NEWS OCTOBER 2017

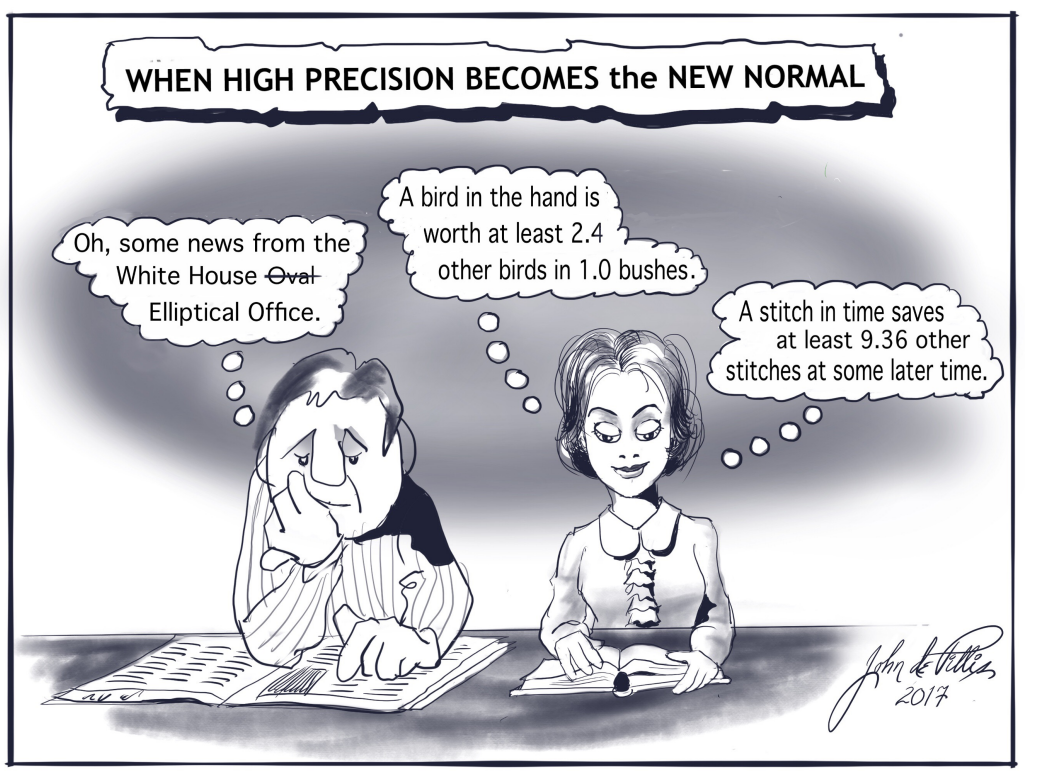
Research | October 02, 2017

Print

A Multiprecision World

By Nicholas Higham

Traditionally, floating-point arithmetic has come in two precisions: single and double. But with the introduction of support for other precisions, thanks in part to the influence of applications, the floating-point landscape has become much richer in recent years.



Floating Point Number System

Floating point number system $F \subset \mathbb{R}$:

$$y = \pm m \times \beta^{e-t}, \qquad 0 \le m \le \beta^t - 1.$$

- *Base* β ($\beta = 2$ in practice),
- *precision* t,
- *exponent range* $e_{\min} \le e \le e_{\max}$.

Assume **normalized**: $m \ge \beta^{t-1}$.

Floating Point Number System

Floating point number system $F \subset \mathbb{R}$:

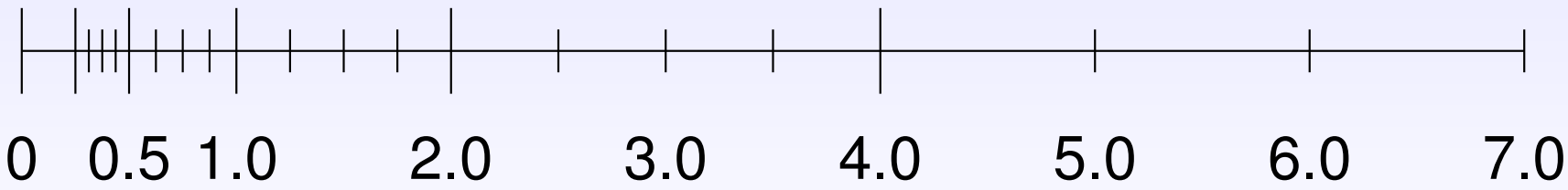
$$y = \pm m \times \beta^{e-t}, \qquad 0 \le m \le \beta^t - 1.$$

- *Base* β ($\beta = 2$ in practice),
- *precision* t,
- *exponent range* $e_{\min} \le e \le e_{\max}$.

Assume **normalized**: $m \ge \beta^{t-1}$.

Floating point numbers are **not equally spaced**.

If $\beta = 2$, $t = 3$, $e_{\min} = -1$, and $e_{\max} = 3$:



Subnormal Numbers

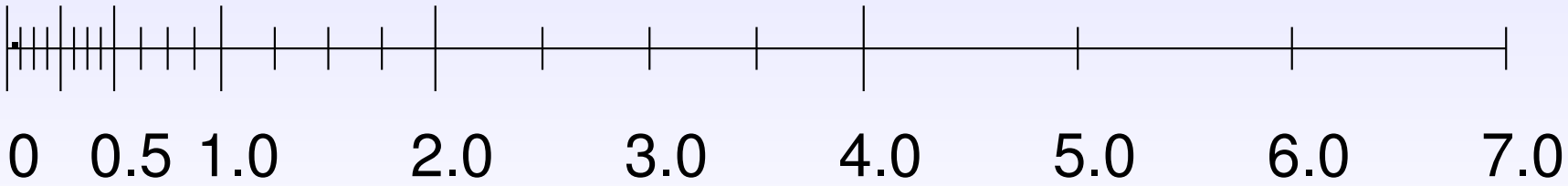
$0 \neq y \in F$ is *normalized* if $m \geq \beta^{t-1}$. Unique representation.

Subnormal numbers have minimum exponent and not normalized:

$$y = \pm m \times \beta^{e_{\min}-t}, \qquad 0 < m < \beta^{t-1},$$

Fewer digits of precision than the normalized numbers.

Subnormal numbers fill the gap between $\beta^{e_{\min}-1}$ and 0 and are *equally spaced*. Including subnormals in our toy system:



IEEE Standard 754-1985 and 2008 Revision

Type	Size	Range	$u = 2^{-t}$
half	16 bits	$10^{\pm 5}$	$2^{-11} \approx 4.9 \times 10^{-4}$
single	32 bits	$10^{\pm 38}$	$2^{-24} \approx 6.0 \times 10^{-8}$
double	64 bits	$10^{\pm 308}$	$2^{-53} \approx 1.1 \times 10^{-16}$
quadruple	128 bits	$10^{\pm 4932}$	$2^{-113} \approx 9.6 \times 10^{-35}$

- Arithmetic ops $(+, -, *, /, \sqrt{})$ performed *as if* first calculated to infinite precision, then rounded.
- Default: round to nearest, round to even in case of tie.
- Half precision is a *storage format only*.

Relative Error

If $\widehat{x} \approx x \in \mathbb{R}^n$ the **relative error** is

$$\frac{\|x - \widehat{x}\|}{\|x\|}.$$

The **absolute error** $\|x - \widehat{x}\|$ is scale dependent.

Common error not to normalize errors and residuals.

Rounding

For $x \in \mathbb{R}$, $fl(x)$ is an element of F nearest to x, and the transformation $x \to fl(x)$ is called **rounding** (to nearest).

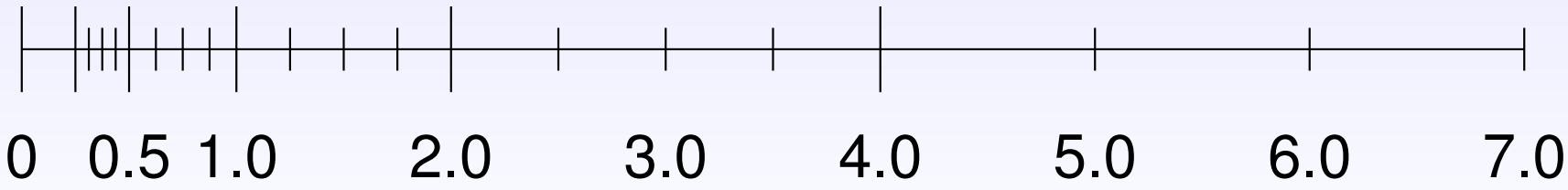
Theorem

If $x \in \mathbb{R}$ lies in the range of F then

$$fl(x) = x(1+\delta), \qquad |\delta| \le u.$$

$u := \frac{1}{2}\beta^{1-t}$ is the **unit roundoff**, or machine precision.

The **machine epsilon**, $\epsilon_M = \beta^{1-t}$ is the spacing between 1 and the next larger floating point number (`eps` in MATLAB).



Model vs Correctly Rounded Result

$y = x(1+\delta)$, with $|\delta| \le u$ does not imply $y = fl(x)$.

$\beta = 10$,
$t = 2$

x	y	$\lvert x-y\rvert/x$	$u = \frac{1}{2}10^{1-t}$
9.185	8.7	5.28e-2	5.00e-2
9.185	8.8	4.19e-2	5.00e-2
9.185	8.9	3.10e-2	5.00e-2
9.185	9.0	2.01e-2	5.00e-2
9.185	9.1	9.25e-3	5.00e-2
9.185	9.2	1.63e-3	5.00e-2
9.185	9.3	1.25e-2	5.00e-2
9.185	9.4	2.34e-2	5.00e-2
9.185	9.5	3.43e-2	5.00e-2
9.185	9.6	4.52e-2	5.00e-2
9.185	9.7	5.61e-2	5.00e-2

Model for Rounding Error Analysis

For $x, y \in F$

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \qquad |\delta| \le u, \quad \text{op} = +, -, *, /.$$

Also for $\text{op} = \sqrt{\ }$.

Model for Rounding Error Analysis

For $x, y \in F$

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \qquad |\delta| \le u, \quad \text{op} = +, -, *, /.$$

Also for op $= \sqrt{\ }$.

Sometimes more convenient to use

$$fl(x \text{ op } y) = \frac{x \text{ op } y}{1+\delta}, \qquad |\delta| \le u, \quad \text{op} = +, -, *, /.$$

Model is weaker than $fl(x \text{ op } y)$ being correctly rounded.

Precision versus Accuracy

$$\begin{aligned} fl(abc) &= ab(1+\delta_1)\cdot c(1+\delta_2) \qquad |\delta_i| \le u, \\ &= abc(1+\delta_1)(1+\delta_2) \\ &\approx abc(1+\delta_1+\delta_2). \end{aligned}$$

- Precision $= u$.
- Accuracy $\approx 2u$.

Precision versus Accuracy

$$\begin{aligned} fl(abc) &= ab(1+\delta_1)\cdot c(1+\delta_2) \qquad |\delta_i| \le u, \\ &= abc(1+\delta_1)(1+\delta_2) \\ &\approx abc(1+\delta_1+\delta_2). \end{aligned}$$

- Precision $= u$.
- Accuracy $\approx 2u$.

Accuracy is not limited by precision


65
220

Fused Multiply-Add Instruction

A multiply-add instruction with just one rounding error:

$$fl(x + y * z) = (x + y * z)(1 + \delta), \qquad |\delta| \leq u.$$

With an FMA:

- Inner product $x^T y$ can be computed with half the rounding errors.
- In the IEEE 2008 standard.
- Supported by much hardware, including NVIDIA Volta architecture (P100, V100) at FP16.

Fused Multiply-Add Instruction (cont.)

- The algorithm of **Kahan**

$$
\begin{aligned}
1 \quad & w = b * c \\
2 \quad & e = w - b * c \\
3 \quad & x = (a * d - w) + e
\end{aligned}
$$

computes $x = \det(\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right])$ with high relative accuracy

But

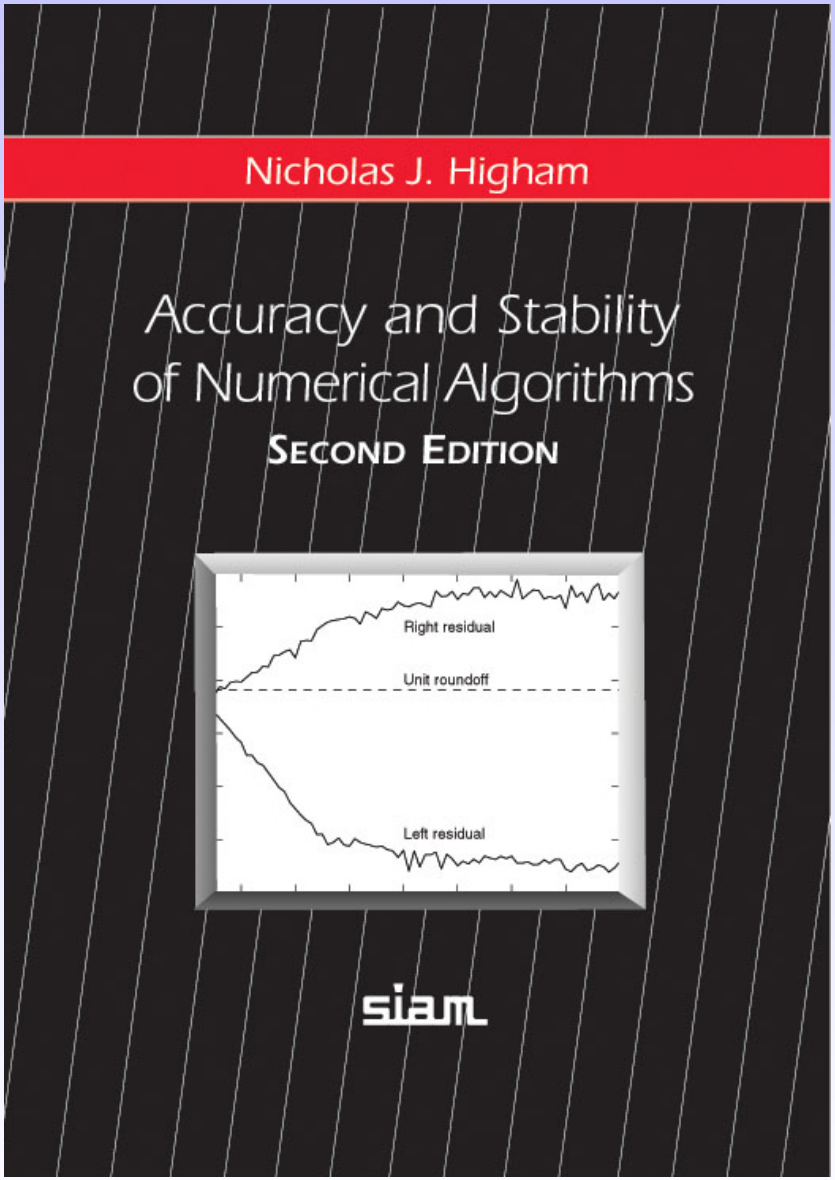
- What does `a*d + c*b` mean?
- The product

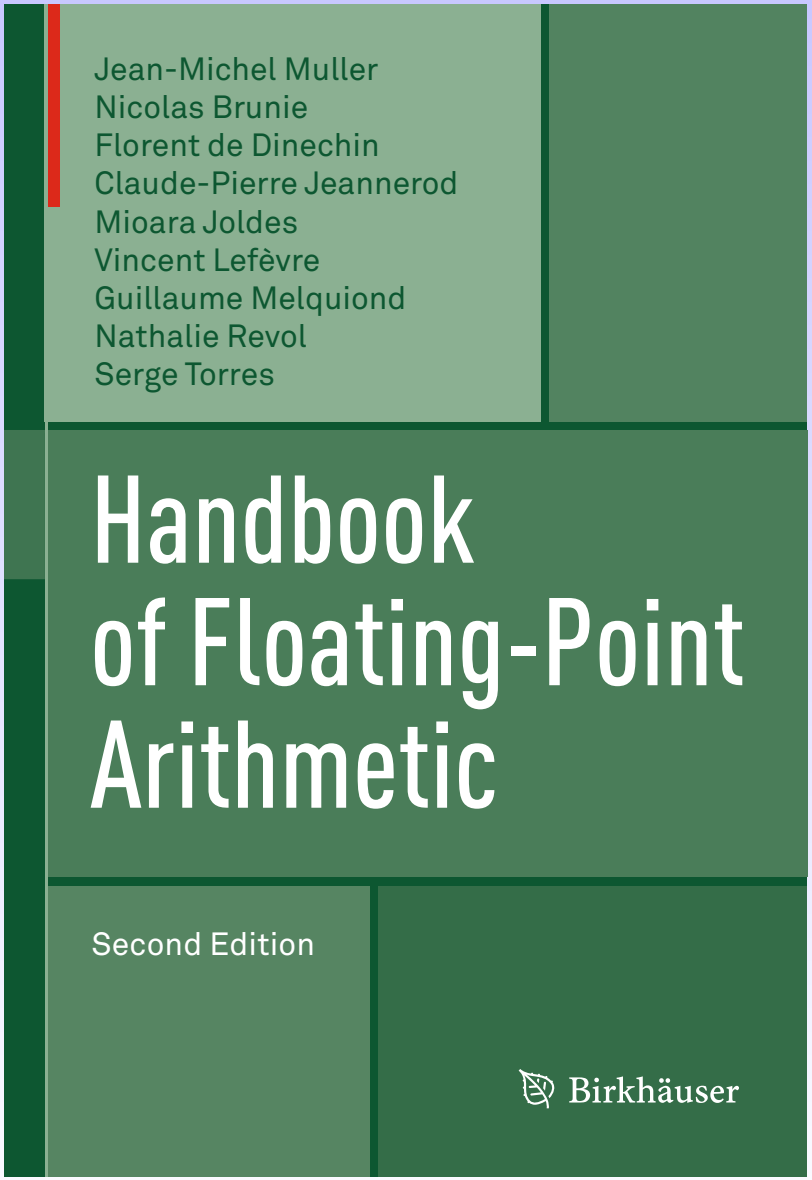
$$(x + iy)^*(x + iy) = x^2 + y^2 + i(xy - yx)$$

may evaluate to non-real with an FMA.

- $b^2 - 4ac$ can evaluate negative even when $b^2 \geq 4ac$.

References for Floating-Point





ARM NEON

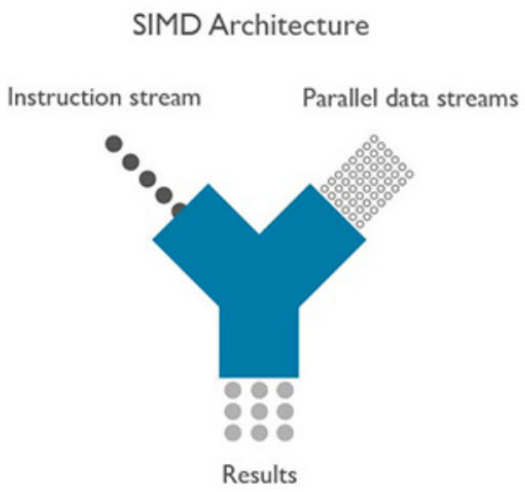
ARM Developer ▾ Products Markets Technologies Support Se

Overview big.LITTLE NEON DSP ▾ DynamIQ Graphics Technologies ▾ TrustZone ▾ Compute Library Floating Point

Overview

The NEON technology is a packed SIMD architecture. NEON registers are considered as vectors of elements of the same data type. Multiple data types are supported by the technology. The following table describes data types as supported by the architecture version.

	ARMv7-A/R	ARMv8-A/R	ARMv8-A
		AArch32	AArch64
Floating-point	32-bit	16-bit*/32-bit	16-bit*/32-bit/64-bit
Integer	8-bit/16-bit/32-bit	8-bit/16-bit/32-bit/64-bit	8-bit/16-bit/32-bit/64-bit

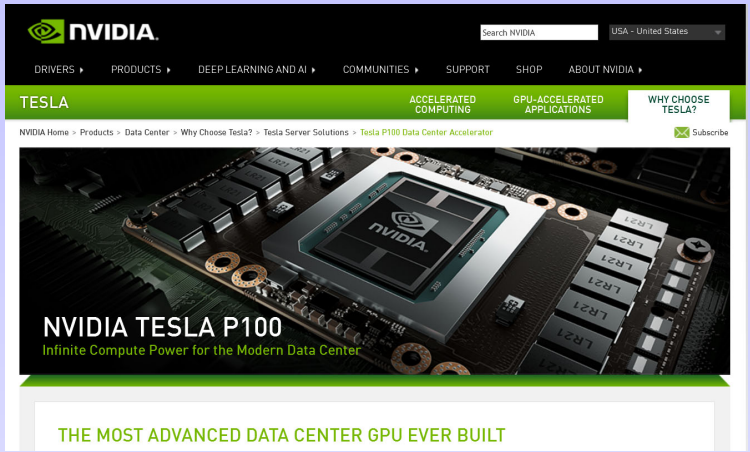


The NEON instructions perform the same operations in all lanes of the vectors. The number of operations performed depends on the data types. NEON instructions allow up to:

- 16x8-bit, 8x16-bit, 4x32-bit, 2x64-bit integer operations
- 8x16-bit*, 4x32-bit, 2x64-bit** floating-point operations

The implementation on NEON technology can also support issue of multiple instructions in parallel.

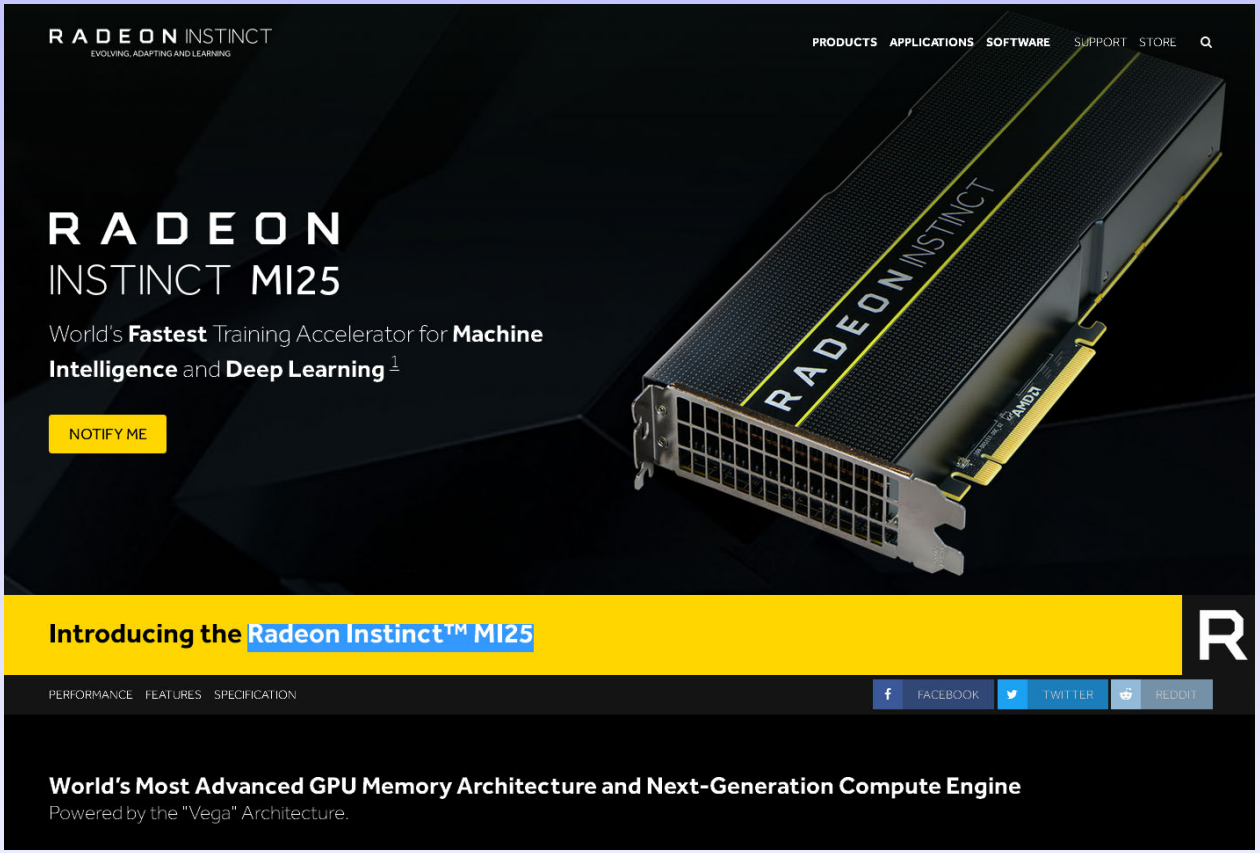
NVIDIA Tesla P100 (2016), V100 (2017)



- "The Tesla P100 is the world's first accelerator built for deep learning, and has native hardware ISA support for **FP16** arithmetic"
- V100 tensor cores do 4×4 mat mult in one clock cycle.

	TFLOPS		
	double	single	half/ tensor
P100	4.7	9.3	18.7
V100	7	14	112

AMD Radeon Instinct MI25 GPU (2017)



"24.6 TFLOPS FP16 or 12.3 TFLOPS FP32 peak GPU compute performance on a single board . . . Up to 82 GFLOPS/watt FP16 or 41 GFLOPS/watt FP32 peak GPU compute performance"

Low Precision in Machine Learning

Widespread use of low precision, for training *and* inference:

single precision (fp32)	32 bits
half precision (fp16)	16 bits
integer (INT8)	8 bits
ternary	$\{-1, 0, 1\}$
binary	$\{0, 1\}$

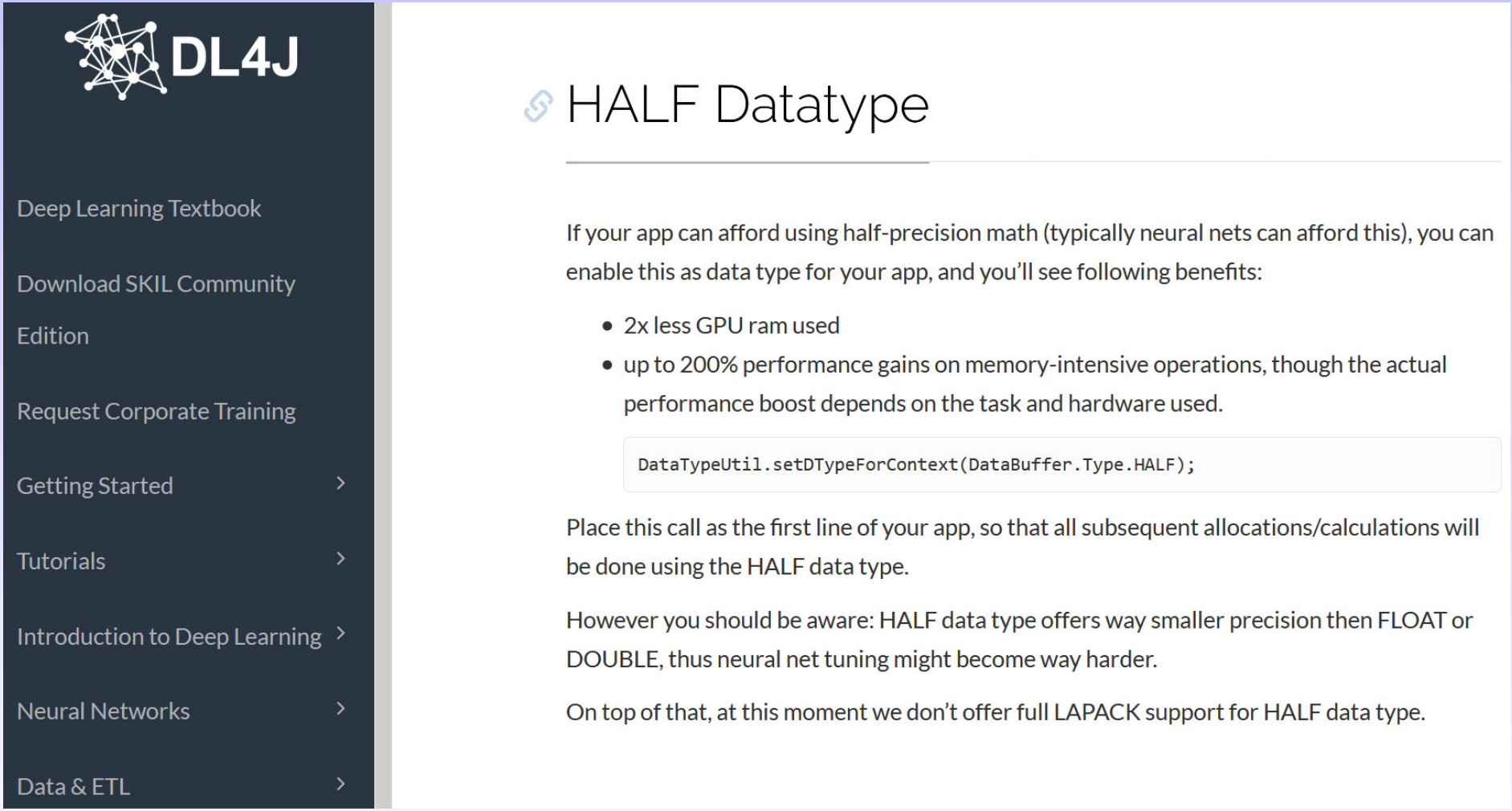
plus other newly-proposed floating-point formats.

- "We find that very low precision is sufficient not just for running trained networks but also for training them." **Courbariaux, Benji & David** (2015)
- No rigorous rounding error analysis exists (yet).
- Papers usually experimental, using particular data sets.

Why Does Low Precision Work in ML?

- We're solving the wrong problem (**Scheinberg**, 2016), so don't need an accurate solution.
- Low precision provides regularization.
- Low precision encourages flat minima to be found.

Deep Learning for Java



Climate Modelling

- **T. Palmer**, **More reliable forecasts with less precise computations: a fast-track route to cloud-resolved weather and climate simulators?**, Phil. Trans. R. Soc. A, 2014:

  > "Is there merit in representing variables at sufficiently high wavenumbers using half or even quarter precision floating-point numbers?"

- **T. Palmer**, **Build imprecise supercomputers**, Nature, 2015.

Fp16 for Communication Reduction

ResNet-50 training on ImageNet.

- Solved in **60 mins on 256 TESLA P100s** at Facebook (2017).
- Solved in **15 mins on 1024 TESLA P100s** at Preferred Networks, Inc. (2017) using ChainerMN (**Takuya Akiba**, SIAM PP18):

Fp16 for Communication Reduction

ResNet-50 training on ImageNet.

- Solved in **60 mins on 256 TESLA P100s** at Facebook (2017).
- Solved in **15 mins on 1024 TESLA P100s** at Preferred Networks, Inc. (2017) using ChainerMN (**Takuya Akiba**, SIAM PP18):
  "While **computation was generally done in single precision, in order to reduce the communication overhead** during all-reduce operations, **we used half-precision floats** . . . In our preliminary experiments, we observed that the effect from using half-precision in communication on the final model accuracy was relatively small."

Preconditioning with Adaptive Precision

Anzt, Dongarra, Flegar, H & Quintana-Ortí (2018):

- For sparse A and iterative $Ax = b$ solver, execution time and energy dominated by **data movement**.
- Block Jacobi preconditioning: $D = \text{diag}(D_i)$, where $D_i = A_{ii}$. Solve $D^{-1}Ax = D^{-1}b$.
- All computations are at **fp64**.
- Compute D^{-1} and store D_i^{-1} in **fp16, fp32** or **fp64**, depending on $\kappa(D_i)$.
- Simulations and energy modelling show can outperform fixed precision preconditioner.

Range Parameters

$r_{\min}^{(s)}$ = *smallest* positive (subnormal) number,
$r_{\min}$ = *smallest normalized* positive number,
$r_{\max}$ = *largest* finite number.

	$r_{\min}^{(s)}$	$r_{\min}$	$r_{\max}$
fp16	5.96×10^{-8}	6.10×10^{-5}	65504
fp32	1.40×10^{-45}	1.18×10^{-38}	3.4×10^{38}
fp64	4.94×10^{-324}	2.22×10^{-308}	1.80×10^{308}

Example: Vector 2-Norm in fp16

Evaluate $\|x\|_2$ for

$$x = \begin{bmatrix} \alpha \\ \alpha \end{bmatrix}$$

as $\sqrt{x_1^2 + x_2^2}$ in fp16.

Recall $u_h = 4.88 \times 10^{-4}$, $r_{\min} = 6.10 \times 10^{-5}$.

α	Relative error	Comment
10^{-4}	1	Underflow to 0
3.3×10^{-4}	4.7×10^{-2}	Subnormal range.
5.5×10^{-4}	7.1×10^{-3}	Subnormal range.
1.1×10^{-2}	1.4×10^{-4}	Perfect rel. err.

A Simple Loop

```
x = pi; i = 0;
while x/2 > 0
   x = x/2; i = i+1;
end
for k = 1:i
   x = 2*x;
end
```

Precision	i	$\lvert x - \pi \rvert$
Double	1076	0.858
Single	151	0.858
Half	26	0.858

A Simple Loop

```
x = pi; i = 0;
while x/2 > 0
   x = x/2; i = i+1;
end
for k = 1:i
   x = 2*x;
end
```

Precision	i	$\lvert x - \pi \rvert$
Double	1076	0.858
Single	151	0.858
Half	26	0.858

- Why these large errors?
- Why the same error for each precision?

Error Analysis in Low Precision (1)

For inner product $x^T y$ of n-vectors standard error bound is

$$|\,\mathrm{fl}(x^T y) - x^T y| \le \gamma_n |x|^T |y|, \qquad \gamma_n = \frac{nu}{1-nu}, \quad nu < 1.$$

Can also be written as

$$|\,\mathrm{fl}(x^T y) - x^T y| \le nu|x|^T|y| + O(u^2).$$

In half precision, $u \approx 4.9 \times 10^{-4}$, so $nu = 1$ for $n = 2048$.

Error Analysis in Low Precision (1)

For inner product $x^T y$ of n-vectors standard error bound is

$$|\mathrm{fl}(x^T y) - x^T y| \le \gamma_n |x|^T |y|, \qquad \gamma_n = \frac{nu}{1-nu}, \quad nu < 1.$$

Can also be written as

$$|\mathrm{fl}(x^T y) - x^T y| \le nu|x|^T|y| + O(u^2).$$

In half precision, $u \approx 4.9 \times 10^{-4}$, so $nu = 1$ for $n = 2048$.

What happens when $nu > 1$?

Error Analysis in Low Precision (2)

Rump & Jeannerod (2014) prove that in a number of standard rounding error bounds, $\gamma_n = nu/(1-nu)$ **can be replaced by** nu **provided that round to nearest is used.**

Error Analysis in Low Precision (2)

Rump & Jeannerod (2014) prove that in a number of standard rounding error bounds, $\gamma_n = nu/(1-nu)$ **can be replaced by** nu **provided that round to nearest is used.**

- Analysis nontrivial. Only a few core algs have been analyzed.
- Be'rr bound for $Ax = b$ is now $(3n-2)u + (n^2-n)u^2$ instead of γ_{3n}.
- Cannot replace γ_n by nu in all algs (e.g., pairwise summation).
- Once $nu \geq 1$ bounds cannot guarantee any accuracy, maybe not even a correct exponent!

Simulating fp16 Arithmetic

- **Simulation 1.**
  Converting operands to fp32 or fp64, carry out the operation in fp32 or fp64, then round the result back to fp16.
- **Simulation 2.**
  Scalar fp16 operations as in Simulation 1. Carry out matrix multiplication and matrix factorization in fp32 or fp64 then round the result back to fp16.

MATLAB fp16 Class (Moler)

Cleve Laboratory fp16 class  uses Simulation 2 for `mtimes` (called in `lu`) and `mldivide`.

http://mathworks.com/matlabcentral/fileexchange/59085-cleve-laboratory

```
function z = plus(x,y)
    z = fp16(double(x) + double(y));
end
function z = mtimes(x,y)
    z = fp16(double(x) * double(y));
end
function z = mldivide(x,y)
    z = fp16(double(x) \ double(y));
end
function [L,U,p] = lu(A)
    [L,U,p] = lutx(A);
end
```

Is Simulation 2 Too Accurate?

- For matrix mult, standard error bound is

$$|C - \widehat{C}| \leq \gamma_n |A||B|, \qquad \gamma_n = nu/(1-nu).$$

  Error bound for Simulation 2 has **no n factor**.

- For triangular solves, $Tx = b$, error should be bounded by $\mathrm{cond}(T, x)\gamma_n$, but we actually get **error of order u**.

- Simulation 1 preferable. but too slow unless the problem is fairly small.
- Large operator overloading overheads in any language.

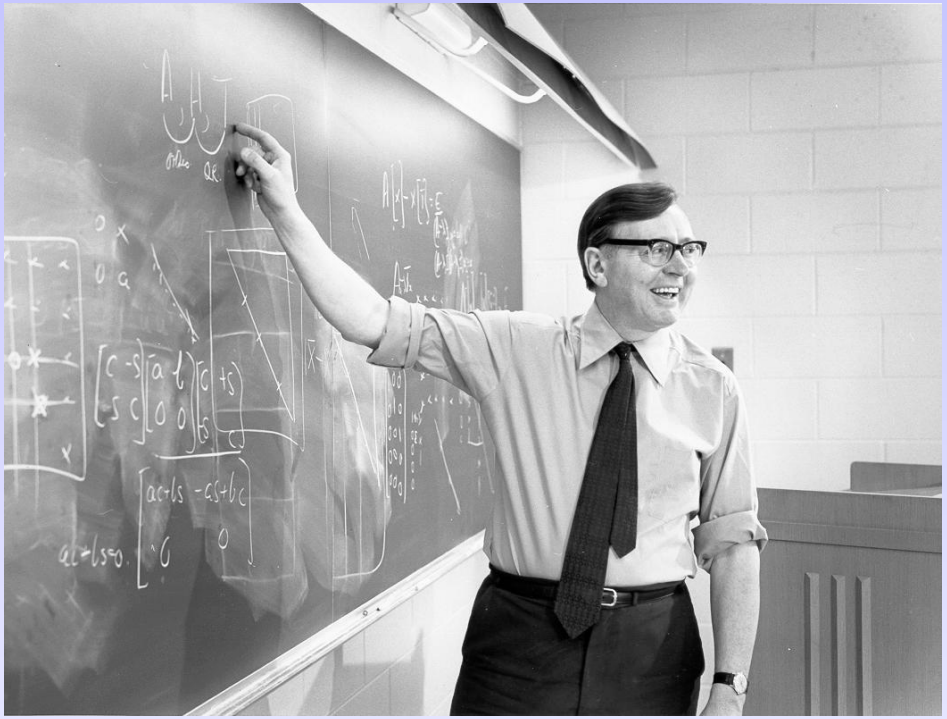
Lecture 2

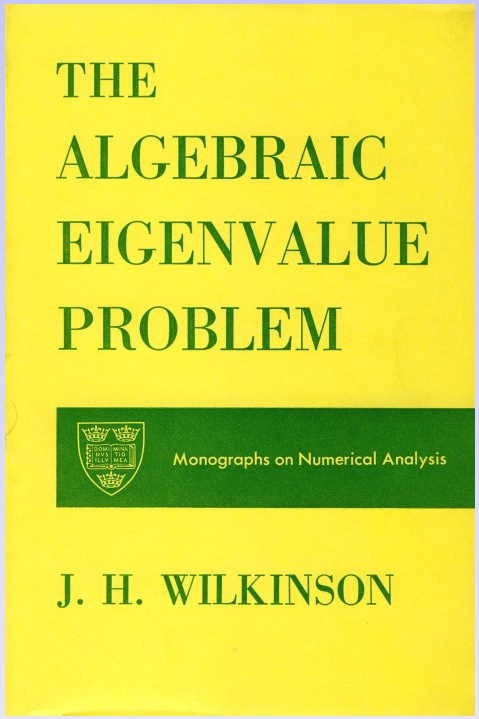
- Rounding error analysis
- Higher precision
- Multiprecision alg for log A.

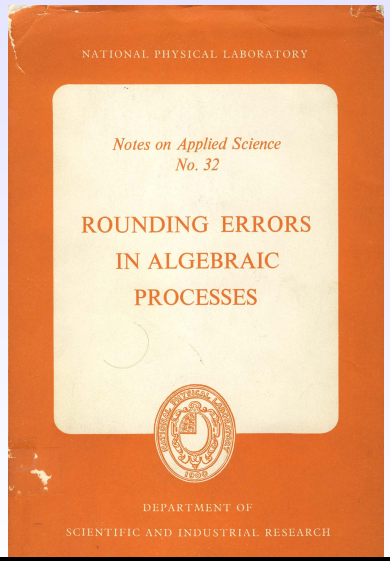
Download these slides from

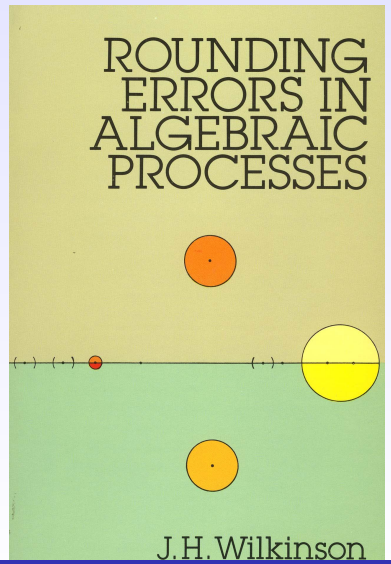
http://bit.ly/kacov18

James Hardy Wilkinson (1919–1986)









(III) DECISIVE.

OBSERVED FOLLOWING PHENOMENON.

FOUND EIGENVALUES & NORMALIZED VECTORS ($\|\bar{x}\|_2 = 1$) BY ITERATION AND DEFLATION.

COMPUTED $r = A\bar{x} - \bar{\lambda}\bar{x}$ $\quad \bar{x}^T\bar{x} = 1$.

OBSERVED THAT r WAS INCREDIBLY SMALL.

$(A - r\bar{x}^T)\bar{x} = \bar{\lambda}\bar{x}$!!

$\bar{\lambda}$ AND $\bar{x}$ EXACT FOR $A - r\bar{x}^T$ AND

IN PRACTICE FOUND THAT

$\|r\bar{x}^T\|_2 = \|r\|$ WAS MODEST MULT.

OF $\beta^{-t}\|A\|$

INCREDIBLY BACKWARD STABLE. WHY

(IV) SUPPORTIVE

OBSERVED FOLLOWING.

SOLVED $Ax = b$ BY GAUS. ELIM

COMPUTED $r = b - A\bar{x}$. NOW r

WAS NOT (IN GENERAL) INCREDIBLY SMALL

BUT. $b = \left(A + \frac{r\bar{x}^T}{\|\bar{x}\|_2^2}\right)\bar{x}$. FOUND THAT

$\left\|\frac{r\bar{x}^T}{\|\bar{x}\|_2^2}\right\| = \frac{\|r\|}{\|\bar{x}\|}$ WAS MODEST MULT OF $\beta^{-t}\|A\|$

INCREDIBLY BACKWARD STABLE. WHY?

Summation

Compute $s_n = \sum_{i=1}^{n} x_i$.

Recursive summation:

```
1  s_1 = x_1
2  for i = 2: n
3        s_i = s_{i-1} + x_i
4  end
```

$$\widehat{s}_2 = fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta_2), \quad |\delta_2| \le u.$$

$$\begin{aligned} \widehat{s}_3 = fl(\widehat{s}_2 + x_3) &= fl(\widehat{s}_2 + x_3)(1 + \delta_3), \quad |\delta_3| \le u \\ &= (x_1 + x_2)(1 + \delta_2)(1 + \delta_3) + x_3(1 + \delta_3). \end{aligned}$$

Etc. . . .

General Summation Algorithm

Algorithm (to compute $s_n = \sum_{i=1}^n x_i$)

1 Let $\mathcal{X} = \{x_1, \ldots, x_n\}$.
2 while $\mathcal{X}$ contains more than one element
3 Remove two numbers y and z from $\mathcal{X}$
and put their sum $y + z$ back in $\mathcal{X}$.
4 end
5 Assign the remaining element of $\mathcal{X}$ to s_n.

Write ith execution of loop as $p_i = y_i + z_i$. Then

$$\widehat{p}_i = \frac{y_i + z_i}{1 + \delta_i}, \quad |\delta_i| \leq u, \quad i = 1\colon n-1,$$

where u is the unit roundoff.

General Summation Algorithm (cont.)

Error in forming $\widehat{p}_i$, namely $y_i + z_i - \widehat{p}_i$, is therefore $\delta_i \widehat{p}_i$. Summing individual errors get overall error

$$e_n := s_n - \widehat{s}_n = \sum_{i=1}^{n-1} \delta_i \widehat{p}_i,$$

which is bounded by

$$|e_n| \leq u \sum_{i=1}^{n-1} |\widehat{p}_i| \,.$$

Bound shows we should **minimize the size of the intermediate partial sums**.

The Difficulty of Rounding Error Analysis

- Deciding what to try to prove.
- Type of analysis:

componentwise	backward error
normwise	forward error
mixed backward–forward error	

- Knowing what model to use.
- Keep or discard second order terms?
- Ignore possibility of underflow and overflow?
- Assume real data?

Multiprecision Rounding Error Analysis

- Increasing motivation to **mix different precisions**: half, single, double, even quarter, quadruple.
- NVIDIA V100 tensor core takes fp16 matrices as input and computes the product using FMAs at fp32; result returned at fp16 or fp32.
- We need **parametrized error analyses** that allow different precisions to be used in different components in an algorithm. *Hard to cover all possibilities!*
- See analysis in third lecture on iterative refinement (3 precisions).

Need for Higher Precision

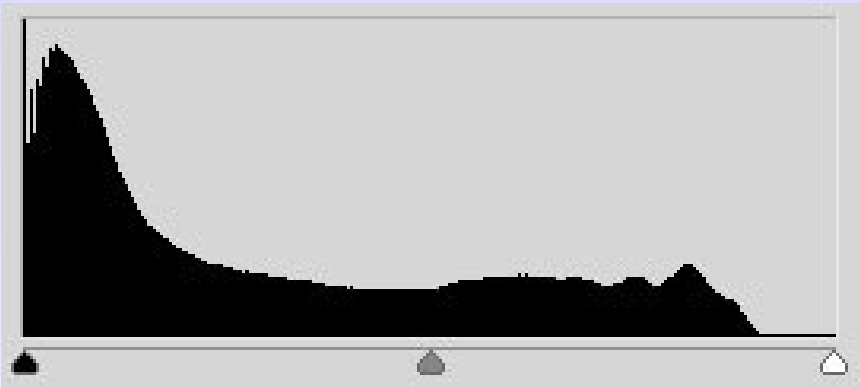
- He and Ding, **Using Accurate Arithmetics to Improve Numerical Reproducibility and Stability in Parallel Applications**, 2001.
- Bailey, Barrio & Borwein, **High-Precision Computation: Mathematical Physics & Dynamics**, 2012.
- Khanna, **High-Precision Numerical Simulations on a CUDA GPU: Kerr Black Hole Tails**, 2013.
- Beliakov and Matiyasevich, **A Parallel Algorithm for Calculation of Determinants and Minors Using Arbitrary Precision Arithmetic**, 2016.
- Ma and Saunders, **Solving Multiscale Linear Programs Using the Simplex Method in Quadruple Precision**, 2015.

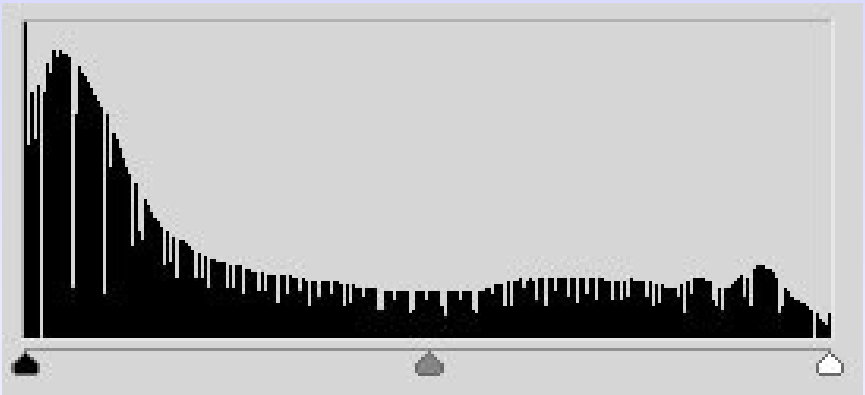
Aside: Rounding Errors in Digital Imaging

- An 8-bit RGB image is an $m \times n \times 3$ array of integers $a_{ijk} \in \{0, 1, \ldots, 255\}$.
- Every editing op (levels, curves, colour balance, ...) $a_{ijk} \leftarrow \text{round}(f_{ijk}(a_{ijk}))$ incurs a rounding error:

Aside: Rounding Errors in Digital Imaging

- An 8-bit RGB image is an $m \times n \times 3$ array of integers $a_{ijk} \in \{0, 1, \ldots, 255\}$.
- Every editing op (levels, curves, colour balance, ...) $a_{ijk} \leftarrow \text{round}(f_{ijk}(a_{ijk}))$ incurs a rounding error:





Should we edit in 16-bit?

16-bit vs. 8-bit editing

- SLR cameras generate image at 12–14 bits internally.
8-bit	$0 : 1 : 255$
16-bit	$0 : 3.9 \times 10^{-3} : 255$

16-bit vs. 8-bit editing

- SLR cameras generate image at 12–14 bits internally.
8-bit	$0 : 1 : 255$
16-bit	$0 : 3.9 \times 10^{-3} : 255$
- **Controversial**: hard to find realistic image where using 16 bits makes any practical difference in quality.

16-bit vs. 8-bit editing

- SLR cameras generate image at 12–14 bits internally.
8-bit	$0 : 1 : 255$
16-bit	$0 : 3.9 \times 10^{-3} : 255$
- **Controversial**: hard to find realistic image where using 16 bits makes any practical difference in quality.

Relevant metric is not normwise relative error!

Increasing the Precision

$y = e^{\pi\sqrt{163}}$ evaluated at t digit precision:

t	y
20	262537412640768744.00
25	262537412640768744.0000000
30	262537412640768743.999999999999

Is the last digit before the decimal point 4?

Increasing the Precision

$y = e^{\pi\sqrt{163}}$ evaluated at t digit precision:

t	y
20	262537412640768744.00
25	262537412640768744.0000000
30	262537412640768743.999999999999

Is the last digit before the decimal point 4?

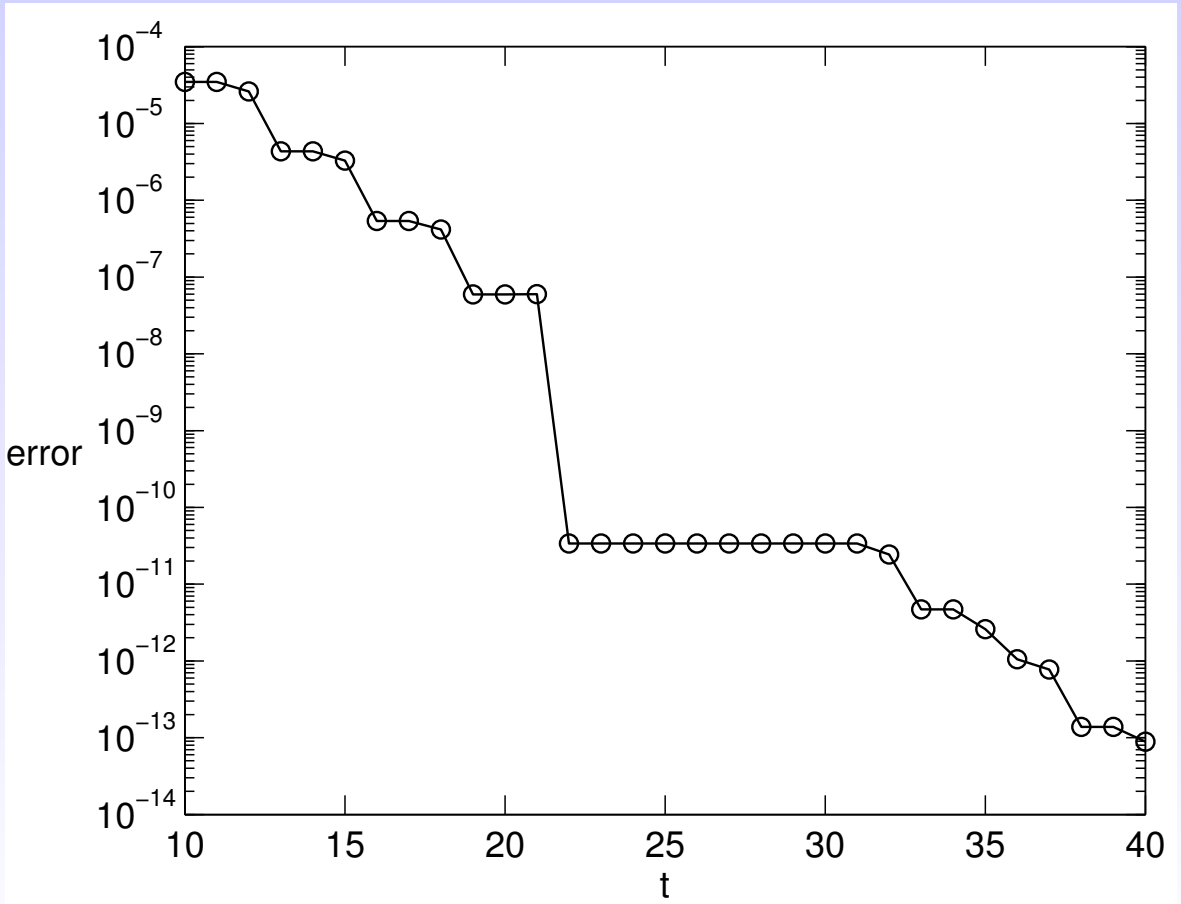
t	y
35	262537412640768743.99999999999925007
40	262537412640768743.9999999999992500725972

So no, it's 3!

Another Example

Consider the evaluation in precision $u = 2^{-t}$ of

$$y = x + a\sin(bx), \quad x = 1/7, \quad a = 10^{-8}, \quad b = 2^{24}.$$



Myth

Increasing the precision at which a computation is performed increases the accuracy of the answer.

Going to Higher Precision

If we have quadruple or higher precision, how can we modify existing algorithms to exploit it?

Going to Higher Precision

If we have quadruple or higher precision, how can we modify existing algorithms to exploit it?

To what extent are existing algs precision-independent?

- Newton-type algs: just decrease `tol`?
- How little higher precision can we get away with?
- Gradually increase precision through the iterations?
- For Krylov methods # iterations can depend on precision, so lower precision might not give fastest computation!

Availability of Multiprecision in Software

- **Maple**, Mathematica, PARI/GP, **Sage**.
- MATLAB: Symbolic Math Toolbox, **Multiprecision Computing Toolbox** (Advanpix).
- Julia: **BigFloat**.
- Mpmath and SymPy for Python.
- GNU MP Library.
- **GNU MPFR Library**.
- (Quad only): some C, Fortran compilers.

Gone, but not forgotten:

- Numerical Turing: **Hull et al.**, 1985.

Cost of Quadruple Precision

How Fast is Quadruple Precision Arithmetic?



Compare

- MATLAB double precision,
- Symbolic Math Toolbox, VPA arithmetic, digits(34),
- Multiprecision Computing Toolbox (Advanpix), mp.Digits(34). Optimized for quad.

Ratios of times

Intel Broadwell-E Core i7-6800K @3.40GHz, 6 cores

	mp/double	vpa/double	vpa/mp
LU, $n = 250$	98	25,000	255
eig, nonsymm, $n = 125$	75	6,020	81
eig, symm, $n = 200$	32	11,100	342

Matrix Functions

(Inverse) scaling and squaring-type algorithms for e^A, $\log A$, $\cos A$, A^t use Padé approximants.

- Padé degree and algorithm parameters chosen to achieve double precision accuracy, $u = 2^{-53}$.
- Change u and the algorithm logic needs changing!
- *Open questions*, even for scalar elementary functions?

Now focus on **matrix logarithm.**

Existing Software

- MATLAB has built-in `expm`, `logm`, `sqrtm`, `funm`. We have written `cosm`, `sinm`, `signm`, `powerm`, `lambertwm`, `unwindm`, ...
- Julia has `expm`, `logm`, `sqrtm`.
- NAG Library has 42+ $f(A)$ codes.

H & Deadman, **A Catalogue of Software for Matrix Functions** (2016). 

Log Tables

Henry Briggs (1561–1630):

- **Arithmetica Logarithmica** (1624)
- Logarithms to base 10 of 1–20,000 and 90,000–100,000 to **14 decimal places**.

Briggs must be viewed as one of the great figures in numerical analysis.

—Herman H. Goldstine,
A History of Numerical Analysis (1977)

Name	Year	Range	Decimal places
R. de Prony	1801	$1 - 10,000$	19
Edward Sang	1875	$1 - 20,000$	28

ARITHMETICA

LOGARITHMICA

SIVE

LOGARITHMORVM CHILIADES TRIGINTA, PRO

numeris naturali ſerie creſcentibus ab vnitate ad 20,000 : et a 90,000 ad 100,000. Quorum ope multa perficiuntur Arithmetica problemata et Geometrica.

HOS NVMEROS PRIMVS INVENIT CLARISSIMVS VIR IOHANNES NEPERVS Baro Merchiſtonij : eos autem ex eiuſdem ſententia mutavit, eorumque ortum et vſum illuſtravit HENRICVS BRIGGIVS, in celeberrima Academia Oxonienſi Geometriæ profeſſor SAVILIANVS.

DEVS NOBIS VSVRAM VITÆ DEDIT ET INGENII, TANQVAM PECVNIÆ, NVLLA PRÆSTITVTA DIE.



LONDINI,
Excudebat GVLIELMVS IONES. 1624.

Numeri continue Medij inter Denarium & Vnitatem.	*Logarithmi Rationales.*
10	1,000
1 31622,77660,16837,93319,98893,54	0,50
2 17782,79410,03892,28011,97304,13	0,25
3 13335,21432,16332,40256,65389,308	0,125
4 11547,81984,68945,81796,61918,213	0,0625
5 10746,07828,32131,74972,13817,6538	0,03125
6 10366,32928,43769,79972,90627,3131	0,01562,5
7 10181,51721,71818,18414,73723,8144	0,00781,25
8 10090,35044,84144,74377,59005,1391	0,00390,625
9 10045,07364,25446,25156,64670,6113	0,00195,3125
10 10022,51148,29291,29154,65611,7367	0,00097,65625
11 10011,24941,39987,98758,85395,52805	0,00048,82812,5
12 10005,62312,60220,86366,18495,91839	0,00024,41406,25
13 10002,81116,78778,01323,99249,64325	0,00012,20703,125
14 10001,40548,51694,72581,62767,32715	0,00006,10351,5625
15 10000,70271,78941,14355,38811,70845	0,00003,05175,78125
16 10000,35135,27746,18565,08581,37077	0,00001,52587,89062,5
17 10000,17567,48442,26738,33846,78274	0,00000,76293,94531,25
18 10000,08783,70363,46121,46574,07431	0,00000,38146,97265,625
19 10000,04391,84217,31672,36281,88083	0,00000,19073,48632,8125
20 10000,02195,91867,55542,03317,07719	0,00000,09536,74316,40625
21 10000,01097,95873,50204,09754,72940	0,00000,04768,37158,20312,5
22 10000,00548,97921,68211,14626,60250,4	0,00000,02384,18579,10156,25
23 10000,00274,48957,07382,95091,25449,9	0,00000,01192,09289,55078,125
24 10000,00137,24477,59510,83282,69572,5	0,00000,00596,04644,77539,0625
25 10000,00068,62238,56210,25737,18748,2	0,00000,00298,02322,38769,53125
26 10000,00034,31119,22218,83912,75020,8	0,00000,00149,01161,19384,76562,5
27 10000,00017,15559,59637,84719,93879,1	0,00000,00074,50580,59692,38281,25
28 10000,00008,57779,79451,03051,17588,8	0,00000,00037,25290,29846,19140,625
29 10000,00004,28889,89633,54198,42901,3	0,00000,00018,62645,14923,09570,3125
30 10000,00002,14444,94793,77767,42970,4	0,00000,00009,31322,57461,54785,15625
31 10000,00001,07222,47391,14050,76926,8	0,00000,00004,65661,28730,77392,57812,5
32 10000,00000,53611,23694,13317,14831,4	0,00000,00002,32830,64365,38696,28906,25
33 10000,00000,26805,61846,70731,51508,7	0,00000,00001,16415,32182,69348,14453,125
34 10000,00000,13402,80923,26383,99277,7	0,00000,00000,58207,66091,34674,07226,5625
35 10000,00000,06701,40461,60946,55519,6	0,00000,00000,29103,83045,67337,03613,28125
36 10000,00000,03350,70230,79911,91730,0	0,00000,00000,14551,91522,83668,51806,64062,5
37 10000,00000,01675,35115,39815,61857,6	0,00000,00000,07275,95761,41834,25903,32031,25
38 10000,00000,00837,67557,69872,72426,9	0,00000,00000,03637,97880,70917,12951,66015,625
39 10000,00000,00418,83778,84927,59087,9	0,00000,00000,01818,98940,35458,56475,83007,8125
40 10000,00000,00209,41889,42461,60262,5	0,00000,00000,00909,49470,17729,28237,91503,90625
41 10000,00000,00104,70944,71230,25311,0	0,00000,00000,00454,74735,08864,64118,95751,95312

The Scalar Logarithm: Comm. ACM

- J. R. Herndon (1961). **Algorithm 48: Logarithm of a complex number**.

The Scalar Logarithm: Comm. ACM

- J. R. Herndon (1961). **Algorithm 48: Logarithm of a complex number**.
- A. P. Relph (1962). **Certification of Algorithm 48: Logarithm of a complex number.**

The Scalar Logarithm: Comm. ACM

- J. R. Herndon (1961). **Algorithm 48: Logarithm of a complex number**.
- A. P. Relph (1962). **Certification of Algorithm 48: Logarithm of a complex number.**
- M. L. Johnson and W. Sangren, W. (1962). **Remark on Algorithm 48: Logarithm of a complex number.**

The Scalar Logarithm: Comm. ACM

- J. R. Herndon (1961). **Algorithm 48: Logarithm of a complex number**.
- A. P. Relph (1962). **Certification of Algorithm 48: Logarithm of a complex number.**
- M. L. Johnson and W. Sangren, W. (1962). **Remark on Algorithm 48: Logarithm of a complex number.**
- D. S. Collens (1964). **Remark on remarks on Algorithm 48: Logarithm of a complex number.**

The Scalar Logarithm: Comm. ACM

- J. R. Herndon (1961). **Algorithm 48: Logarithm of a complex number**.
- A. P. Relph (1962). **Certification of Algorithm 48: Logarithm of a complex number.**
- M. L. Johnson and W. Sangren, W. (1962). **Remark on Algorithm 48: Logarithm of a complex number.**
- D. S. Collens (1964). **Remark on remarks on Algorithm 48: Logarithm of a complex number.**
- D. S. Collens (1964). **Algorithm 243: Logarithm of a complex number: Rewrite of Algorithm 48.**

Principal Logarithm and pth Root

Let $A \in \mathbb{C}^{n\times n}$ have no eigenvalues on $\mathbb{R}^-$.

Principal log

$X = \log A$ denotes unique X such that

- $e^X = A$.
- $-\pi < \mathrm{Im}(\lambda(X)) < \pi$.

The Average Eye

First order character of optical system characterized by **transference matrix**

$$T = \begin{bmatrix} S & \delta \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{5\times 5},$$

where $S \in \mathbb{R}^{4\times 4}$ is **symplectic**:

$$S^T J S = J = \begin{bmatrix} 0 & I_2 \\ -I_2 & 0 \end{bmatrix}.$$

Average $m^{-1}\sum_{i=1}^m T_i$ is not a transference matrix.

Harris (2005) proposes the average $\exp(m^{-1}\sum_{i=1}^m \log T_i)$.

Inverse Scaling and Squaring

$$\log A = \log \left(A^{1/2^s}\right)^{2^s} = 2^s \log \left(A^{1/2^s}\right)$$

Inverse Scaling and Squaring

$$\log A = \log\left(A^{1/2^s}\right)^{2^s} = 2^s \log\left(A^{1/2^s}\right)$$

Algorithm **Transformation-free** inverse scaling & squaring

1: $s \leftarrow 0$

3: **while** $\|A - I\|_2 \geq 1$ **do**
4: $\quad A \leftarrow A^{1/2}$
5: $\quad s \leftarrow s + 1$
6: **end while**
7: Find Padé approximant r_{km} to $\log(1 + x)$
8: **return** $2^s r_{km}(A - I)$

Inverse Scaling and Squaring

$$\log A = \log\left(A^{1/2^s}\right)^{2^s} = 2^s \log\left(A^{1/2^s}\right)$$

Algorithm **Schur–Padé** inverse scaling & squaring

1: $s \leftarrow 0$
2: **Compute the Schur decomposition** $A := QTQ^*$
3: **while** $\|T - I\|_2 \geq 1$ **do**
4: $\quad T \leftarrow T^{1/2}$
5: $\quad s \leftarrow s + 1$
6: **end while**
7: Find Padé approximant r_{km} to $\log(1 + x)$
8: **return** $2^s Q^* r_{km}(T - I)Q$

- What **degree** for the Padé approximants?
- Should we take more **square roots** once $\|T - I\|_2 < 1$?

Bounding the Error

- **Kenney & Laub** (1989) derived *absolute f'err* bound

$$\|\log(I+X) - r_{km}(X)\| \le |\log(1-\|X\|) - r_{km}(-\|X\|)|.$$

- **Al-Mohy, H & Relton** (2012, 2013) derive b'err bound. Strategy requires symbolic and high prec. comp. (done *off-line*) and obtains parameters dependent on u.
- **Fasi & H** (2018) use a *relative f'err* bound based on $\|\log(I+X)\| \approx \|X\|$, since $\|X\| \ll 1$ is possible.

Strategy: Use f'err bound to minimize s subject to f'err bounded by u (arbitrary) for suitable m.

Sharper Error Bound for Nonnormal Matrices

Al-Mohy & H (2009) note that $\left\|\sum_{i=\ell}^{\infty} c_i A^i\right\|_\infty \le \sum_{i=\ell}^{\infty} |c_i| \|A\|^i$ is too weak. Instead can use

$$\|A^4\| \le \|A^2\|^2 = \left(\|A^2\|^{1/2}\right)^4,$$
$$\|A^{12}\| \le \|A^3\|^4 = \left(\|A^3\|^{1/3}\right)^{12}, \ldots$$

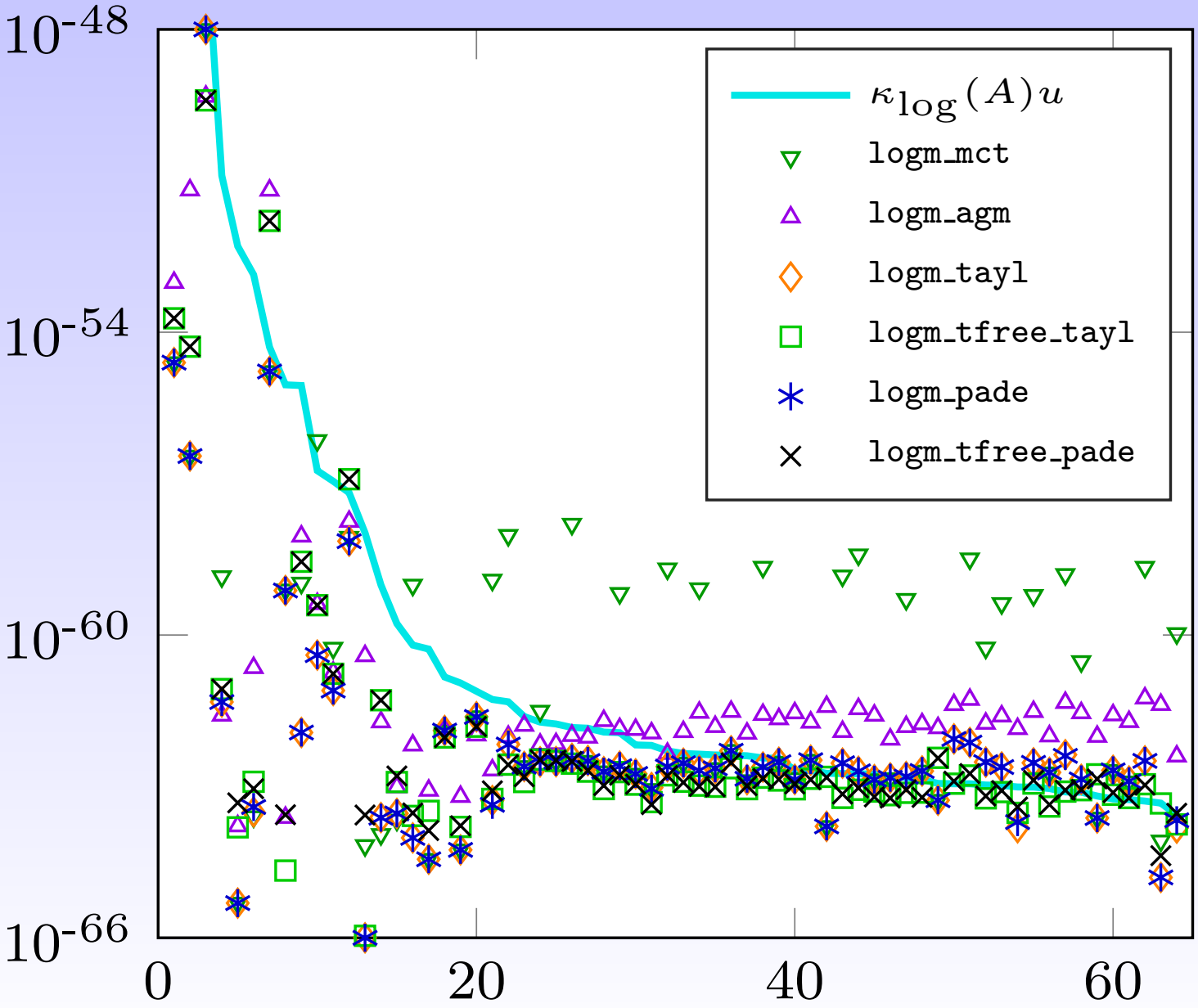
Fasi & H (2018) refine Kenney & Laub's bound using

$$\alpha_p(X) = \max\left(\|X^p\|^{1/p}, \|X^{p+1}\|^{1/(p+1)}\right),$$

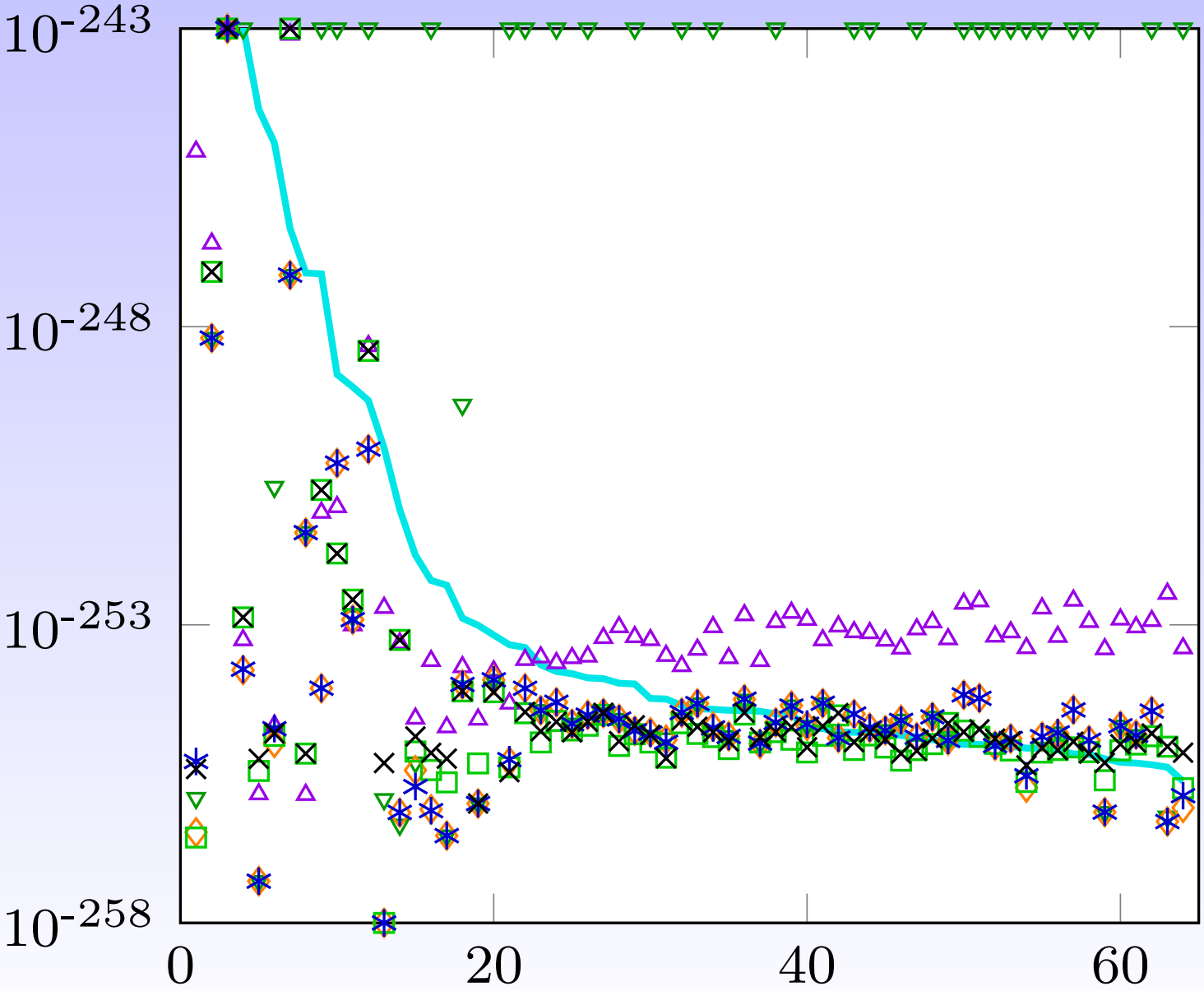
in place of $\|A\|$. Note that $\rho(A) \le \alpha_p(A) \le \|A\|$.

Even sharper bounds derived by **Nadukandi & H** (2018).

64 Digits


10^{-48}
10^{-54}
10^{-60}
10^{-66}
0
20
40
60
$\kappa_{\log}(A)u$
logm_mct
logm_agm
logm_tayl
logm_tfree_tayl
logm_pade
logm_tfree_pade

256 Digits



Conclusions

For matrix log at high precision, since u is now a variable:

- Error bound computed at run-time, not fixed.
- Relative error bound needed.

Tests exposed weakness in existing toolbox at high precisions (since fixed).

Iterative Refinement

Accelerating the Solution of $Ax = b$

$A \in \mathbb{R}^{n\times n}$ nonsingular.

Standard method for solving $Ax = b$: factorize $A = LU$, solve $LUx = b$, all at working precision.

Can we solve $Ax = b$ **faster** and/or **more accurately** by exploiting multiprecision arithmetic?

Iterative Refinement for $Ax = b$ (classic)

Solve $Ax_0 = b$ by LU factorization in **double precision**.

- $r = b - Ax_0$ **quad precision**
- Solve $Ad = r$ **double precision**
- $x_1 = x_0 + d$ **double precision**

($x_0 \leftarrow x_1$ and iterate as necessary.)

- Programmed in **J. H. Wilkinson**, *Progress Report on the Automatic Computing Engine* (1948).
- Popular up to 1970s, exploiting cheap accumulation of inner products.

Iterative Refinement (1970s, 1980s)

Solve $Ax_0 = b$ by LU factorization.

- $r = b - Ax_0$
- Solve $Ad = r$
- $x_1 = x_0 + d$

Everything in **double precision**.

- **Skeel** (1980).
- **Jankowski & Woźniakowski** (1977) for a general solver.

Iterative Refinement (2000s)

Solve $Ax_0 = b$ by LU factorization in **single precision**.

- $r = b - Ax_0$ **double precision**
- Solve $Ad = r$ **single precision**
- $x_1 = x_0 + d$ **double precision**

- **Dongarra et al.** (2006).
- Motivated by single precision being at least twice as fast as double.

Iterative Refinement in Three Precisions

A, b given in **precision u**.

Solve $Ax_0 = b$ by LU factorization in **precision u_f**.

- $r = b - Ax_0$ **precision u_r**
- Solve $Ad = r$ **precision u_f**
- $x_1 = \mathrm{fl}(x_0 + d)$ **precision u**

- Three previous usages are special cases.
- Choose precisions from half, single, double, quadruple subject to $u_r \leq u \leq u_f$.
- **Can we compute *more accurate* solutions *faster*?**

Very Ill-Conditioned Problems

Allow

$$\kappa(A) = \|A\|\|A^{-1}\| \gtrsim u^{-1}.$$

Applications include:

- Reference solutions for testing solvers.
- Radial basis functions.
- Ill-conditioned FE geomechanical problems.

Very Ill-Conditioned Problems

Allow

$$\kappa(A) = \|A\|\|A^{-1}\| \gtrsim u^{-1}.$$

Applications include:

- Reference solutions for testing solvers.
- Radial basis functions.
- Ill-conditioned FE geomechanical problems.

- Base precision may be half or single.
- In low precision *almost every problem is ill conditioned!*

Existing Rounding Error Analysis

- **Wilkinson** (1963): fixed-point arithmetic.
- **Moler** (1967): floating-point arithmetic.
- **Higham** (1997, 2002): more general analysis for arbitrary solver.
- **Dongarra et al.** (2006): lower precision LU.

At most two precisions and require $\kappa(A)u < 1$.

New Analysis

- Applies to any solver.
- Covers b'err and f'err. Focus mainly on f'err.
- Allows $\kappa(A)u \gtrsim 1$.

New Analysis

Assume computed solution to $Ad_i = r_i$ has normwise rel b'err $O(u_f)$ and satisfies

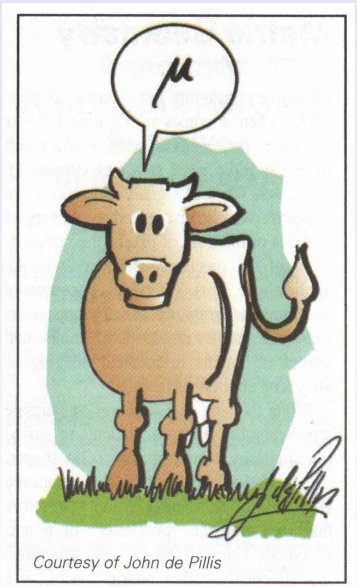
$$\frac{\|d_i - \widehat{d}_i\|_\infty}{\|d_i\|} \leq u_f\,\theta_i < 1.$$

Define μ_i by

$$\|A(x - \widehat{x}_i)\|_\infty = \mu_i \|A\|_\infty \|x - \widehat{x}_i\|_\infty,$$

and note that

$$\kappa_\infty(A)^{-1} \leq \mu_i \leq 1.$$



Condition Numbers

$|A| = (|a_{ij}|)$.

$$\operatorname{cond}(A, x) = \frac{\| \, |A^{-1}||A||x| \, \|_\infty}{\|x\|_\infty},$$

$$\operatorname{cond}(A) = \operatorname{cond}(A, e) = \| \, |A^{-1}||A| \, \|_\infty,$$

$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty.$$

$$1 \le \operatorname{cond}(A, x) \le \operatorname{cond}(A) \le \kappa_\infty(A).$$

Convergence Result

Theorem (Carson & H, 2018)

For IR in precisions $u_r \le u \le u_f$ if

$$\phi_i = 2u_f \min\bigl(\mathrm{cond}(A), \kappa_\infty(A)\mu_i\bigr) + u_f\theta_i$$

is sufficiently less than 1*, the forward error is reduced on the ith iteration by a factor $\approx \phi_i$ until an iterate $\widehat{x}$ satisfies*

$$\frac{\|x - \widehat{x}\|_\infty}{\|x\|_\infty} \lesssim 4nu_r\mathrm{cond}(A,x) + u.$$

Analogous standard bound would have

- $\mu_i = 1$,
- $u_f\theta_i = \kappa(A)u_f$.

Backward Error Result

Theorem (Carson & H, 2018)

For IR in precisions $u_r \le u \le u_f$, if

$$\phi = (c_1 \kappa_\infty(A) + c_2) u_f$$

is sufficiently less than 1 *then the residual is reduced on each iteration by a factor approximately ϕ until*

$$\|b - A\widehat{x}_i\|_\infty \lesssim nu(\|b\|_\infty + \|A\|_\infty \|\widehat{x}_i\|_\infty).$$

Precision Combinations

H = half, S = single, D = double, Q = quad. "u_f u u_r":

Traditional:

SSD
DDQ
HHS
HHD
HHQ
SSQ

1970s/1980s:

SSS
DDD
HHH
QQQ

2000s:

SDD
HSS
DQQ
HDD
HQQ
SQQ

3 precisions:

HSD
HSQ
HDQ
SDQ

Results for LU Factorization (1)

u_f	u	u_r	$\kappa_\infty(A)$	Backward error norm	Backward error comp	Forward error
H	S	S	10^4	S	S	cond$(A,x)\cdot$S
H	S	D	10^4	S	S	S
H	D	D	10^4	D	D	cond$(A,x)\cdot$D
H	D	Q	10^4	D	D	D
S	S	S	10^8	S	S	cond$(A,x)\cdot$S
S	S	D	10^8	S	S	S
S	D	D	10^8	D	D	cond$(A,x)\cdot$D
S	D	Q	10^8	D	D	D

Results (2): HSD vs. SSD

u_f	u	u_r	$\kappa_\infty(A)$	Backward error norm	Backward error comp	Forward error
H	S	S	10^4	S	S	cond$(A,x)\cdot$S
H	S	D	10^4	S	S	S
H	D	D	10^4	D	D	cond$(A,x)\cdot$D
H	D	Q	10^4	D	D	D
S	S	S	10^8	S	S	cond$(A,x)\cdot$S
S	S	D	10^8	S	S	S
S	D	D	10^8	D	D	cond$(A,x)\cdot$D
S	D	Q	10^8	D	D	D

Results (2): HSD vs. SSD

u_f	u	u_r	$\kappa_\infty(A)$	Backward error norm	Backward error comp	Forward error
H	S	S	10^4	S	S	$\mathrm{cond}(A,x)\cdot$ S
H	S	D	10^4	S	S	S
H	D	D	10^4	D	D	$\mathrm{cond}(A,x)\cdot$ D
H	D	Q	10^4	D	D	D
S	S	S	10^8	S	S	$\mathrm{cond}(A,x)\cdot$ S
S	S	D	10^8	S	S	S
S	D	D	10^8	D	D	$\mathrm{cond}(A,x)\cdot$ D
S	D	Q	10^8	D	D	D

Can we get the benefit of "HSD" while allowing a larger range of $\kappa_\infty(A)$?

Extending the Range of Applicability

Recall that the convergence condition is

$$\phi_i = 2u_f \min\bigl(\mathrm{cond}(A), \kappa_\infty(A)\mu_i\bigr) + u_f\,\theta_i \ll 1.$$

- We need **both terms** to be smaller than $\kappa_\infty(A)u_f$.

Recall that

$$\frac{\|d_i - \widehat{d}_i\|_\infty}{\|d_i\|} \leq u_f\,\theta_i,$$

$$\mu_i \|A\|_\infty \|x - x_i\|_\infty = \|A(x - x_i)\|_\infty = \|b - Ax_i\|_\infty = \|r_i\|_\infty.$$

Bounding μ_i (1)

For the 2-norm, can show that

$$\mu_i \le \frac{\|r_i\|_2}{\|P_k r_i\|_2} \frac{\sigma_{n+1-k}}{\sigma_1},$$

where $A = U\Sigma V^T$ is an SVD, $P_k = U_k U_k^T$ with $U_k = [u_{n+1-k}, \dots, u_n]$.

Expect $\mu_i \ll 1$ when r_i contains a significant component in the subspace span(U_k) for any k such that $\sigma_{n+1-k} \approx \sigma_n$ (e.g., $k = 1$).

Bounding μ_i (2)

For a stable solver, in the early stages we expect

$$\frac{\|r_i\|}{\|A\|\|x_i\|} \approx u \ll \frac{\|x - x_i\|}{\|x\|},$$

or equivalently $\mu_i \ll 1$. But **close to convergence**

$$\frac{\|r_i\|}{\|A\|\|x_i\|} \approx u \approx \frac{\|x - x_i\|}{\|x\|} \quad \text{or} \quad \mu_i \approx 1.$$

Conclude

$\mu_i \ll 1$ initially and $\mu_i \to 1$ as the iteration converges.

Bounding θ_i

- $u_f\,\theta_i$ bounds rel error in solution of $Ad_i = r_i$.
- We need $u_f\,\theta_i \ll 1$.

Standard solvers cannot achieve this for very ill conditioned A!

Empirically observed by **Rump (1990)** that if $\widehat{L}$ and $\widehat{U}$ are computed LU factors of A from GEPP then

$$\kappa(\widehat{L}^{-1}A\widehat{U}^{-1}) \approx 1 + \kappa(A)u,$$

even for $\kappa(A) \gg u^{-1}$.

Preconditioned GMRES

To compute the updates d_i we apply **GMRES** to

$$\widetilde{A}d_i \equiv \widehat{U}^{-1}\widehat{L}^{-1}Ad_i = \widehat{U}^{-1}\widehat{L}^{-1}r_i.$$

- $\widetilde{A}$ is applied in twice the working precision.
- $\kappa(\widetilde{A}) \ll \kappa(A)$ typically.
- Rounding error analysis shows we get an accurate $\widehat{d}_i$ even for numerically singular A.
- Call the overall alg **GMRES-IR**.

- GMRES cgce rate not directly related to $\kappa(\widetilde{A})$.

Cf. **Kobayashi & Ogita** (2015), who explicitly form $\widetilde{A}$.

Benefits of GMRES-IR

Recall $H = 10^{-4}$, $S = 10^{-8}$, $D = 10^{-16}$, $Q = 10^{-34}$.

	u_f	u	u_r	$\kappa_\infty(A)$	Backward error nrm	cmp	F'error
LU	H	D	Q	10^4	D	D	D
LU	S	D	Q	10^8	D	D	D

Benefits of GMRES-IR

Recall $H = 10^{-4}$, $S = 10^{-8}$, $D = 10^{-16}$, $Q = 10^{-34}$.

	u_f	u	u_r	$\kappa_\infty(A)$	Backward error nrm	cmp	F'error
LU	H	D	Q	10^4	D	D	D
LU	S	D	Q	10^8	D	D	D
GMRES-IR	H	D	Q	10^{12}	D	D	D
GMRES-IR	S	D	Q	10^{16}	D	D	D

Benefits of GMRES-IR

Recall $H = 10^{-4}$, $S = 10^{-8}$, $D = 10^{-16}$, $Q = 10^{-34}$.

	u_f	u	u_r	$\kappa_\infty(A)$	Backward error nrm	cmp	F'error
LU	H	D	Q	10^4	D	D	D
LU	S	D	Q	10^8	D	D	D
GMRES-IR	H	D	Q	10^{12}	D	D	D
GMRES-IR	S	D	Q	10^{16}	D	D	D

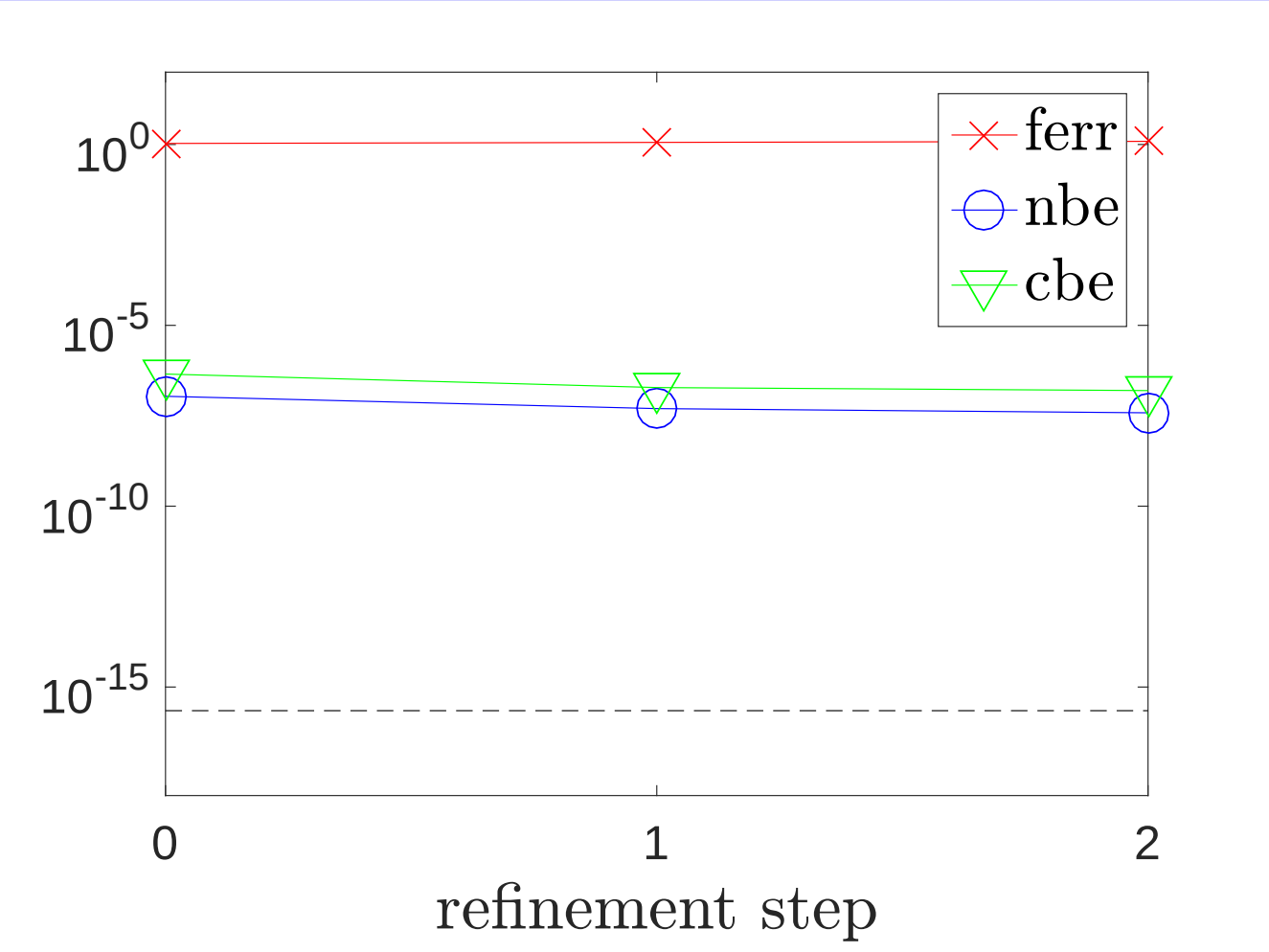
How many GMRES iterations are required?

Some tests with 100×100 matrices ...

Test 1: LU-IR, $(u_f, u, u_r) = (S, D, D)$

$\kappa_\infty(A) \approx 10^{10}$, $\sigma_i = \alpha^i$

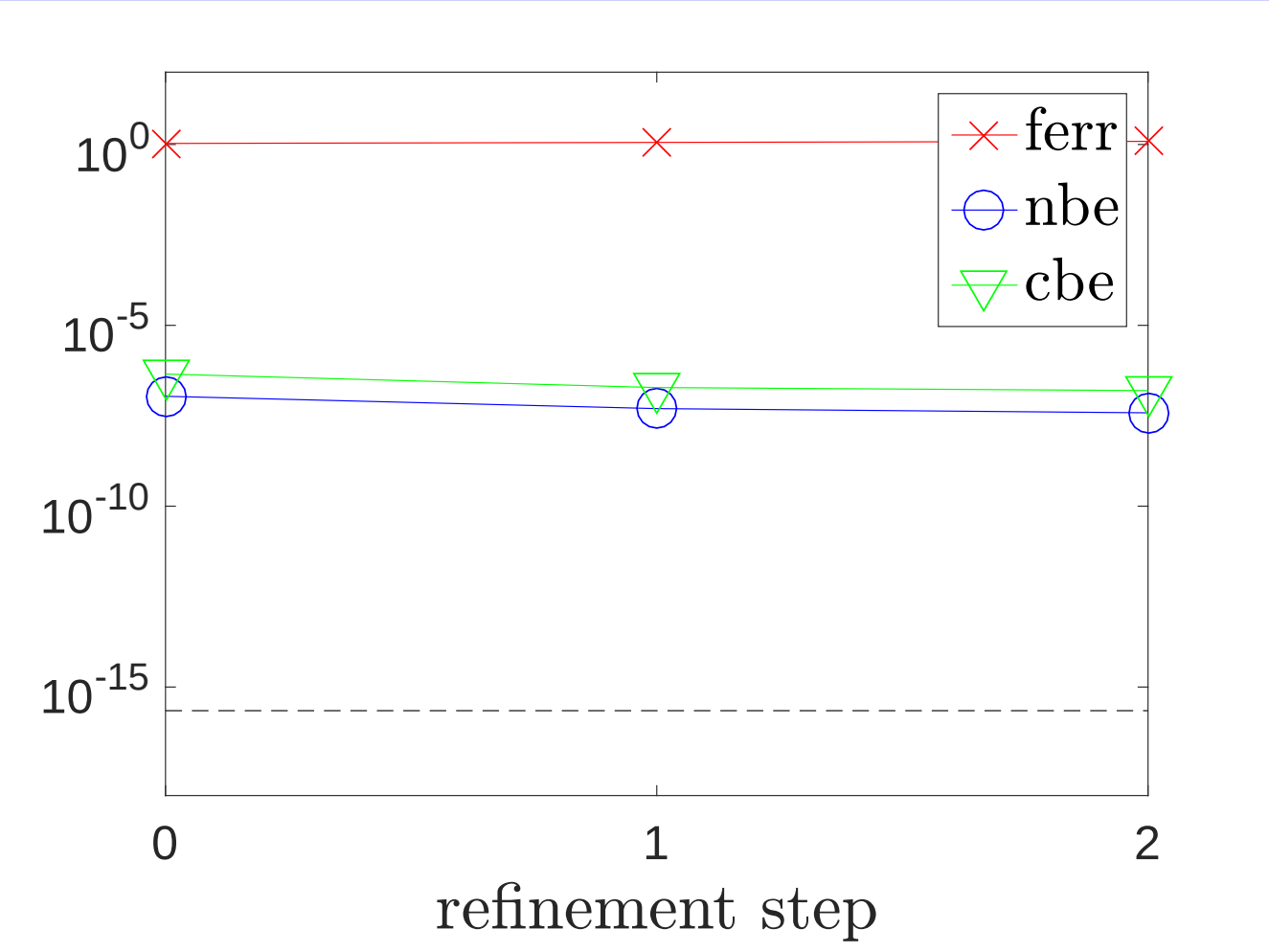
Divergence!



Test 1: LU-IR, $(u_f, u, u_r) = (S, D, Q)$

$\kappa_\infty(A) \approx 10^{10}$, $\sigma_i = \alpha^i$

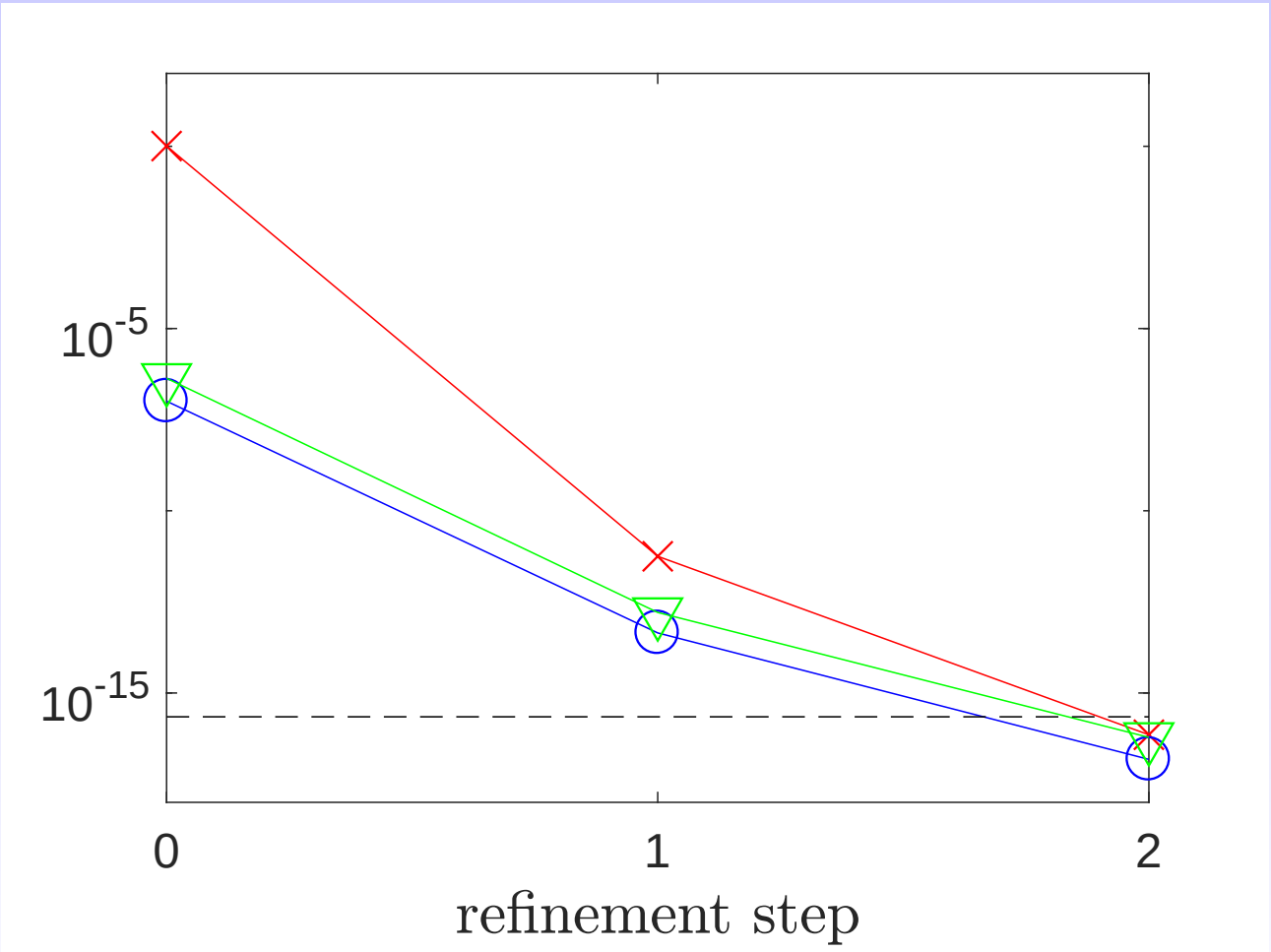
Divergence!



Test 1: LU-IR, $(u_f, u, u_r) = (S, D, Q)$

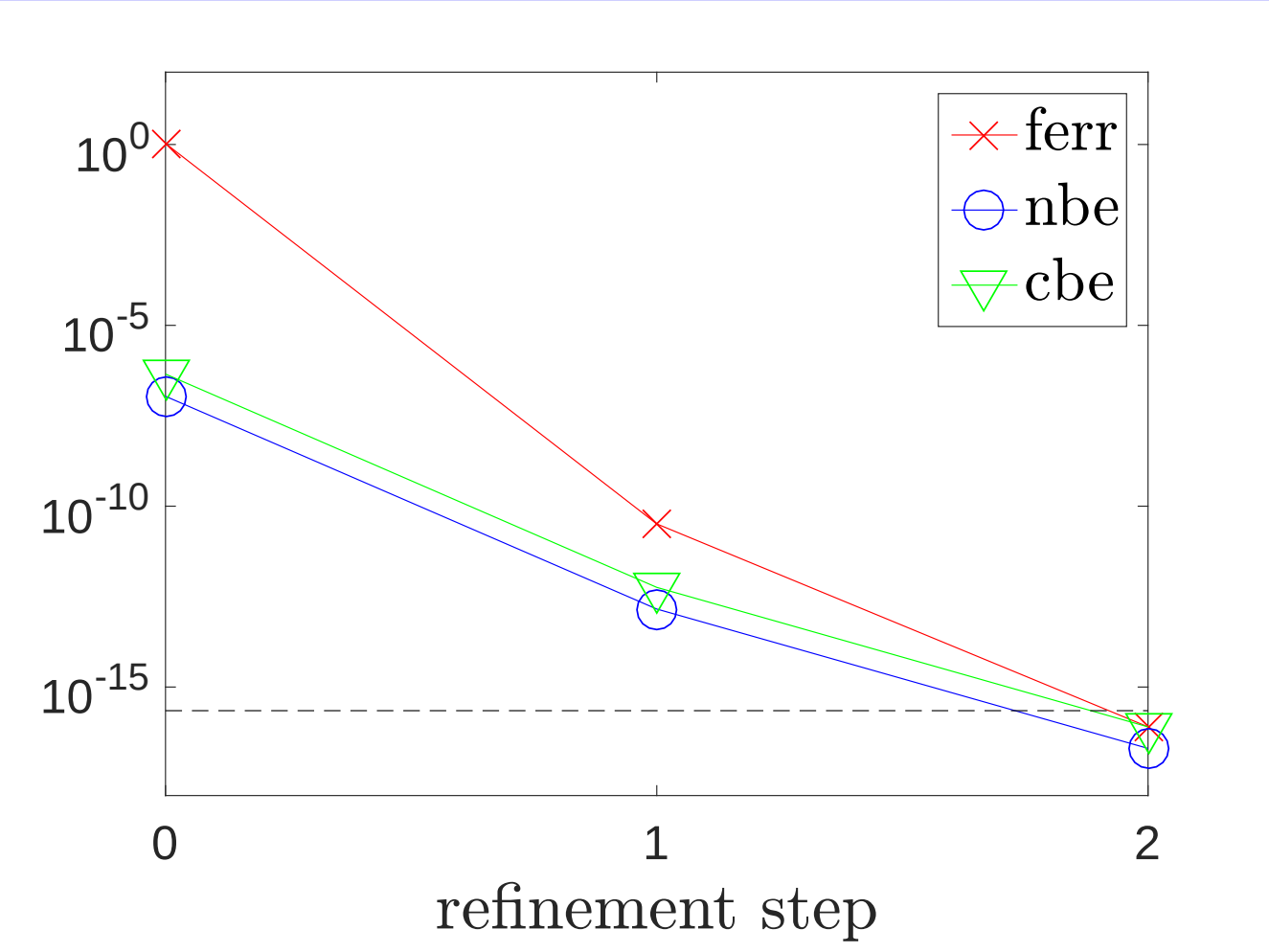
$\kappa_\infty(A) \approx 10^4$, $\sigma_i = \alpha^i$

Convergence



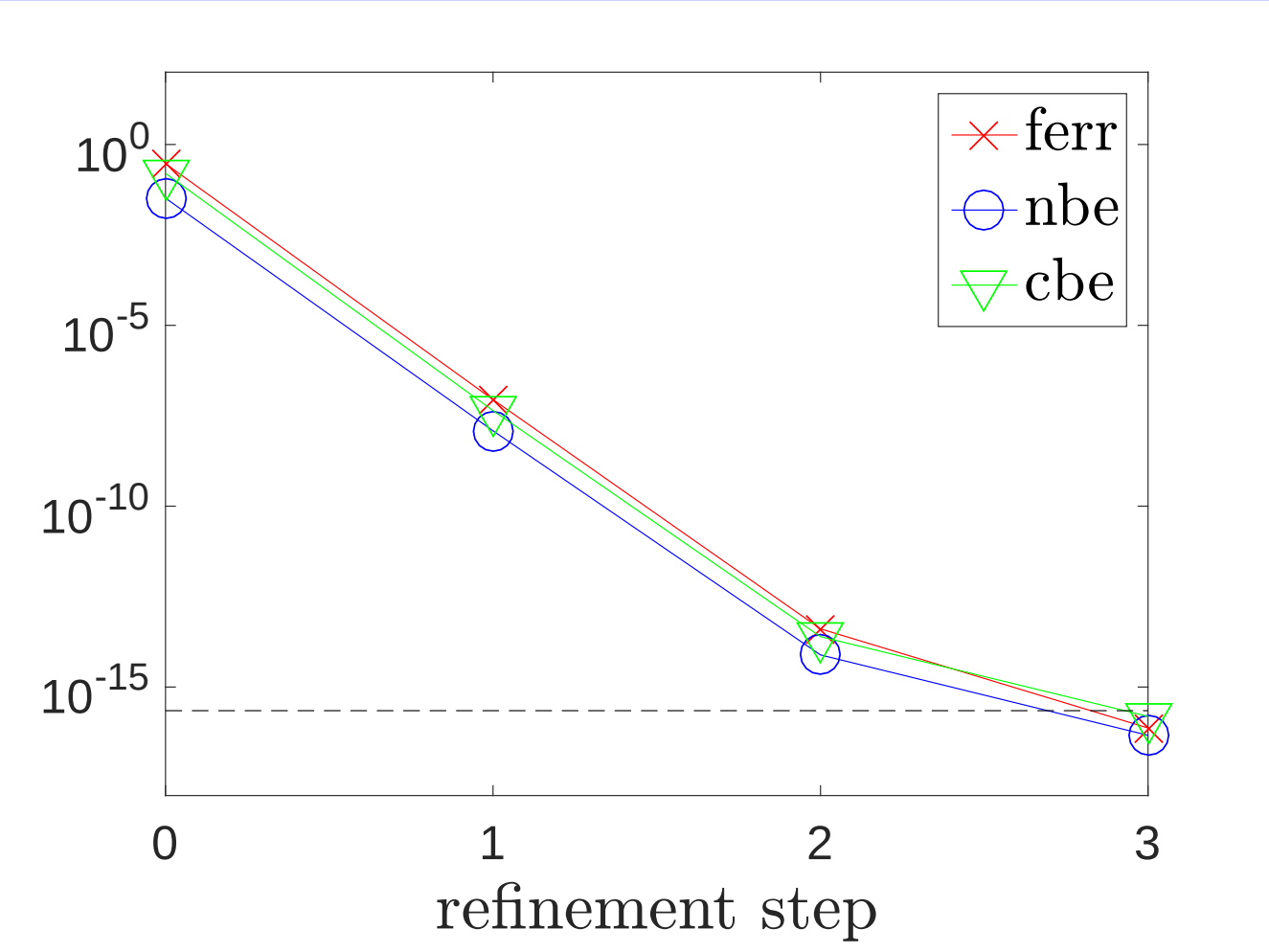
Test 1: GMRES-IR, $(u_f, u, u_r) = (S, D, Q)$

$\kappa_\infty(A) \approx 10^{10}$, $\sigma_i = \alpha^i$, GMRES its (2,3) **Convergence**



Test 2: GMRES-IR, $(u_f, u, u_r) = (H, D, Q)$

$\kappa_\infty(A) \approx 10^2$, 1 small σ_i, GMRES its (8,8,8) **Convergence**

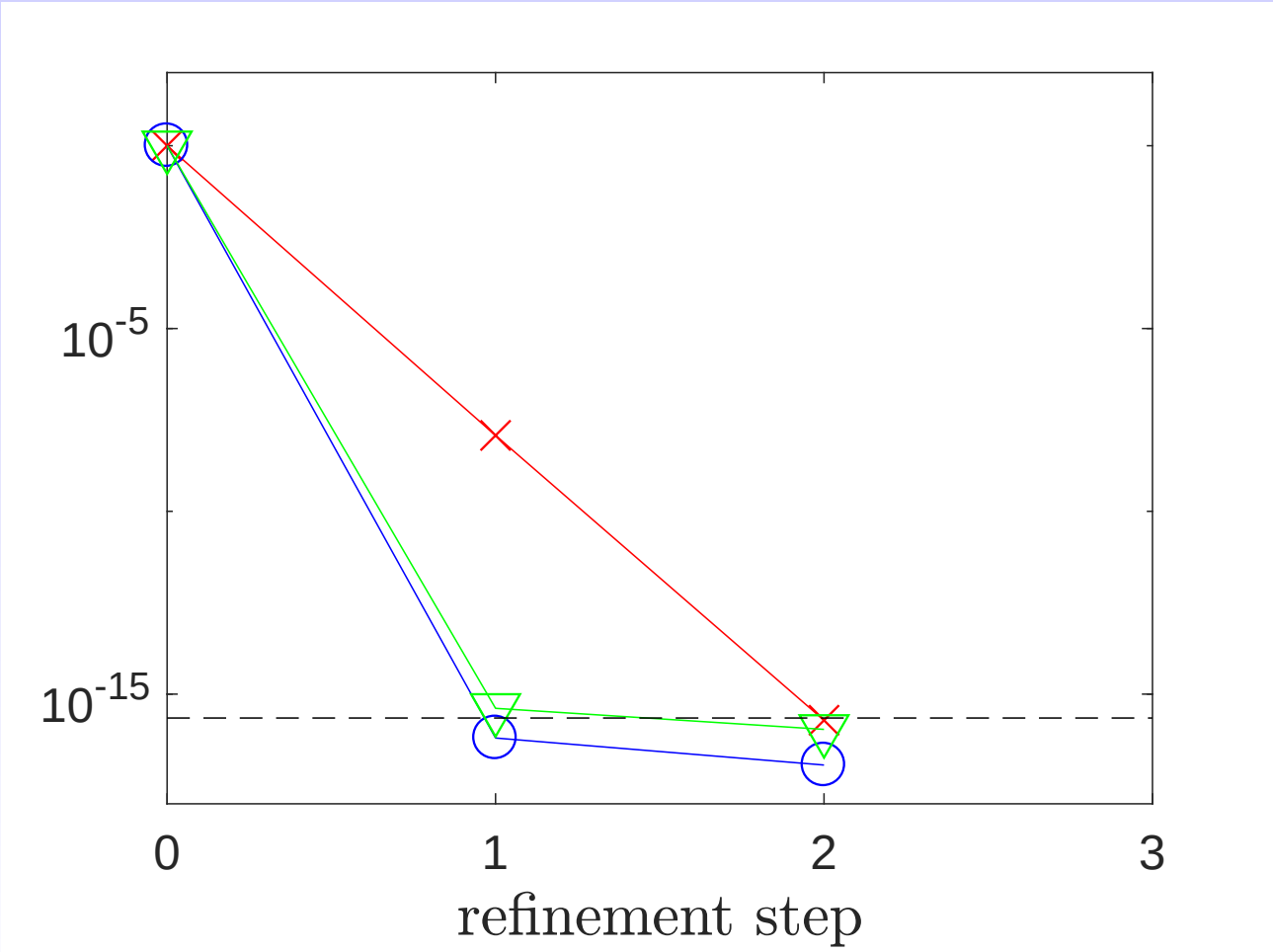


Test 3: GMRES-IR, $(u_f, u, u_r) = (H, D, Q)$

$\kappa_\infty(A) \approx 10^{12}$, $\sigma_i = \alpha^i$, GMRES (100,100)

Take $x_0 = 0$ because of overflow!

Convergence



Other Mixed Precision Work

- **Yamazaki, Tomov & Dongarra (2015)**: Compute QR fact of A via Cholesky of A^TA, using twice working precision to overcome instability for large $\kappa(A)$.
- **Petschow, Quintana-Ortí & Bientinesi (2004)**: multiple relatively robust representations (MRRR) for symm tridiagonal eigenproblem. Extra precision provides improved accuracy.
- **Bini and Robol (2014)**: multiprecision alg for polynomial rootfinding. fp64 + GNU MP Library.

Conclusions (1)

- Both **low and high precision** floating-point arithmetic becoming more prevalent, in hardware and software.
- Need **better understanding** of behaviour of algs in low precision arithmetic.
- Judicious use of **a little high precision** can bring major benefits.

Conclusions (2)

- Showed that using **three precisions** in iter ref brings major benefits.
  - LU in lower precision $\Rightarrow$ twice as fast as trad. IR.
  - GMRES-IR cges when trad. IR doesn't thanks to preconditioned GMRES solved of $Ad_i = r_i$.
  - **GMRES-IR** handles $\kappa_\infty(A) \approx u^{-1}$. Further work: tune cgce tol, alternative preconditioners etc.
- Matrix functions at high precision need algs with (at least) different logic.

How to Do Research

- Develop new metric for solved problem then develop a new alg (or tweak an existing one) so it does best on that metric. E.g. measure communcation costs as well as arithmetic.

How to Do Research

- Develop new metric for solved problem then develop a new alg (or tweak an existing one) so it does best on that metric. E.g. measure communcation costs as well as arithmetic.
- Find hidden assumptions in existing method and remove them. E.g.: Strassen realized use of inner products was not necessary for matrix multiplication.

How to Do Research

- Develop new metric for solved problem then develop a new alg (or tweak an existing one) so it does best on that metric. E.g. measure communcation costs as well as arithmetic.
- Find hidden assumptions in existing method and remove them. E.g.: Strassen realized use of inner products was not necessary for matrix multiplication.
- Find simpler proof of existing result.

How to Do Research

- Develop new metric for solved problem then develop a new alg (or tweak an existing one) so it does best on that metric. E.g. measure communcation costs as well as arithmetic.
- Find hidden assumptions in existing method and remove them. E.g.: Strassen realized use of inner products was not necessary for matrix multiplication.
- Find simpler proof of existing result.
- Prove (weaker version of) existing result under weaker assumptions.

How to Do Research

- Develop new metric for solved problem then develop a new alg (or tweak an existing one) so it does best on that metric. E.g. measure communcation costs as well as arithmetic.
- Find hidden assumptions in existing method and remove them. E.g.: Strassen realized use of inner products was not necessary for matrix multiplication.
- Find simpler proof of existing result.
- Prove (weaker version of) existing result under weaker assumptions.
- Develop alg that is faster and more accurate than existing ones.

How to Do Research

- Develop new metric for solved problem then develop a new alg (or tweak an existing one) so it does best on that metric. E.g. measure communcation costs as well as arithmetic.
- Find hidden assumptions in existing method and remove them. E.g.: Strassen realized use of inner products was not necessary for matrix multiplication.
- Find simpler proof of existing result.
- Prove (weaker version of) existing result under weaker assumptions.
- Develop alg that is faster and more accurate than existing ones.
- Develop alg that can solve problems other algs can't (e.g., GMRES-IR for $\kappa(A) \gg u^{-1}$).

References I


H. Anzt, J. Dongarra, G. Flegar, N. J. Higham, and E. S. Quintana-Ortí.
Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers.
Concurrency Computat.: Pract. Exper., 2018.


D. H. Bailey, R. Barrio, and J. M. Borwein.
High-precision computation: Mathematical physics and dynamics.
Appl. Math. Comput., 218(20):10106–10121, 2012.


G. Beliakov and Y. Matiyasevich.
A parallel algorithm for calculation of determinants and minors using arbitrary precision arithmetic.
BIT, 56(1):33–50, 2016.

References II


D. A. Bini and L. Robol.
Solving secular and polynomial equations: A multiprecision algorithm.
J. Comput. Appl. Math., 272:276–292, 2014.


E. Carson and N. J. Higham.
A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems.
SIAM J. Sci. Comput., 39(6):A2834–A2856, 2017.


E. Carson and N. J. Higham.
Accelerating the solution of linear systems by iterative refinement in three precisions.
SIAM J. Sci. Comput., 40(2):A817–A847, 2018.

References III


M. Courbariaux, Y. Bengio, and J.-P. David.
Training deep neural networks with low precision multiplications, 2015.
ArXiv preprint 1412.7024v5.


M. Fasi and N. J. Higham.
Multiprecision algorithms for computing the matrix logarithm.
SIAM J. Matrix Anal. Appl., 39(1):472–491, 2018.


W. F. Harris.
The average eye.
Opthal. Physiol. Opt., 24:580–585, 2005.

References IV


Y. He and C. H. Q. Ding.
Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications.
J. Supercomputing, 18(3):259–277, 2001.


N. J. Higham.
Iterative refinement for linear systems and LAPACK.
IMA J. Numer. Anal., 17(4):495–509, 1997.


N. J. Higham.
Accuracy and Stability of Numerical Algorithms.
Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.
ISBN 0-89871-521-0.
xxx+680 pp.

References V



G. Khanna.
High-precision numerical simulations on a CUDA GPU: Kerr black hole tails.
J. Sci. Comput., 56(2):366–380, 2013.



Y. Kobayashi and T. Ogita.
A fast and efficient algorithm for solving ill-conditioned linear systems.
JSIAM Lett., 7:1–4, 2015.

References VI



J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and J. Dongarra.
Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems).
In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, Nov. 2006.



D. Ma and M. Saunders.
Solving multiscale linear programs using the simplex method in quadruple precision.
In M. Al-Baali, L. Grandinetti, and A. Purnama, editors, *Numerical Analysis and Optimization*, number 134 in Springer Proceedings in Mathematics and Statistics, pages 223–235. Springer-Verlag, Berlin, 2015.

References VII



S. Markidis, W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter.
NVIDIA tensor core programmability, performance & precision.
ArXiv preprint 1803.04014, Mar. 2018.



J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres.
Handbook of Floating-Point Arithmetic.
Birkhäuser, Boston, MA, USA, second edition, 2018.
ISBN 978-3-319-76525-9.
xxv+627 pp.

References VIII



P. Nadukandi and N. J. Higham.
Computing the wave-kernel matrix functions.
MIMS EPrint 2018.4, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Feb. 2018.
23 pp.



M. Petschow, E. Quintana-Ortí, and P. Bientinesi.
Improved accuracy and parallelism for MRRR-based eigensolvers—A mixed precision approach.
SIAM J. Sci. Comput., 36(2):C240–C263, 2014.

References IX



A. Roldao-Lopes, A. Shahzad, G. A. Constantinides, and E. C. Kerrigan.
More flops or more precision? Accuracy parameterizable linear equation solvers for model predictive control.
In *17th IEEE Symposium on Field Programmable Custom Computing Machines*, pages 209–216, Apr. 2009.



S. M. Rump and C.-P. Jeannerod.
Improved backward error bounds for LU and Cholesky factorizations.
SIAM J. Matrix Anal. Appl., 35(2):684–698, 2014.

References X



K. Scheinberg.
Evolution of randomness in optimization methods for supervised machine learning.
SIAG/OPT Views and News, 24(1):1–8, 2016.



I. Yamazaki, S. Tomov, and J. Dongarra.
Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs.
SIAM J. Sci. Comput., 37(3):C307–C330, 2015.