Multiprecision Algorithms

Nick Higham
School of Mathematics
The University of Manchester

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

Samuel D. Conte Distinguished Lecture, Department of Computer Science, Purdue University, March 2018
Early textbook cmpwise backw error analysis of LU factorization: computed solution of $Ax = b$ satisfies

$$(A + \Delta A)\hat{x} = b, \quad |\Delta A| \leq u_n|A| + u_n(3 + u_n)|\hat{L}|\hat{U}|,$$

where $u_n = 1.01nu$ and $|A| = (|a_{ij}|)$. 
Multiprecision arithmetic: floating point arithmetic supporting multiple precisions.

- **Low precision**—half, or less.
- **High precision**—quadruple, possibly arbitrary.
- How to exploit different precisions to achieve faster algs with higher accuracy.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range</th>
<th>$u = 2^{-t}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>half</td>
<td>16 bits</td>
<td>$10^{\pm5}$</td>
<td>$2^{-11} \approx 4.9 \times 10^{-4}$</td>
</tr>
<tr>
<td>single</td>
<td>32 bits</td>
<td>$10^{\pm38}$</td>
<td>$2^{-24} \approx 6.0 \times 10^{-8}$</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>$10^{\pm308}$</td>
<td>$2^{-53} \approx 1.1 \times 10^{-16}$</td>
</tr>
<tr>
<td>quadruple</td>
<td>128 bits</td>
<td>$10^{\pm4932}$</td>
<td>$2^{-113} \approx 9.6 \times 10^{-35}$</td>
</tr>
</tbody>
</table>

- Arithmetic ops (+, −, *, /, √) 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.
For $x, y \in F$

$$fl(x \text{ op } y) = (x \text{ op } y)(1 + \delta), \quad |\delta| \leq u, \quad \text{op} = +, -, *, /. \quad \text{Also for op} = \sqrt{\cdot}.$$
Precision versus Accuracy

\[ fl(abc) = ab(1 + \delta_1) \cdot c(1 + \delta_2) \quad |\delta_i| \leq u, \]
\[ = abc(1 + \delta_1)(1 + \delta_2) \]
\[ \approx abc(1 + \delta_1 + \delta_2). \]

- **Precision** = \( u \).
- **Accuracy** \( \approx 2u \).
Precision versus Accuracy

\[ fl(abc) = ab(1 + \delta_1) \cdot c(1 + \delta_2) \quad |\delta_i| \leq u, \]
\[ = abc(1 + \delta_1)(1 + \delta_2) \]
\[ \approx abc(1 + \delta_1 + \delta_2). \]

- Precision = \(u\).
- Accuracy \(\approx 2u\).

Accuracy is not limited by precision.
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.

<table>
<thead>
<tr>
<th></th>
<th>ARMv7-A/R</th>
<th>ARMv8-A/R</th>
<th>ARMv8-A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating-point</td>
<td>32-bit</td>
<td>16-bit*/32-bit</td>
<td>16-bit*/32-bit/64-bit</td>
</tr>
<tr>
<td>Integer</td>
<td>8-bit/16-bit/32-bit</td>
<td>8-bit/16-bit/32-bit/64-bit</td>
<td>8-bit/16-bit/32-bit/64-bit</td>
</tr>
</tbody>
</table>

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.
“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 $4 \times 4$ mat mult in one clock cycle.

<table>
<thead>
<tr>
<th></th>
<th>TFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>double</td>
</tr>
<tr>
<td>P100</td>
<td>4.7</td>
</tr>
<tr>
<td>V100</td>
<td>7</td>
</tr>
</tbody>
</table>
“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”
“For machine learning as well as for certain image processing and signal processing applications, more data at lower precision actually yields better results with certain algorithms than a smaller amount of more precise data.”

“We find that very low precision is sufficient not just for running trained networks but also for training them.”

Courbariaux, Benji & David (2015)

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.
HALF Datatype

If your app can afford using half-precision math (typically neural nets can afford this), you can enable this as data type for your app, and you'll see following benefits:

- 2x less GPU ram used
- up to 200% performance gains on memory-intensive operations, though the actual performance boost depends on the task and hardware used.

```java
DataTypeUtil.setTypeForContext(DataBuffer.Type.HALF);
```

Place this call as the first line of your app, so that all subsequent allocations/calculations will be done using the HALF data type.

However you should be aware: HALF data type offers way smaller precision then FLOAT or DOUBLE, thus neural net tuning might become way harder.

On top of that, at this moment we don't offer full LAPACK support for HALF data type.
- 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?

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.
Error Analysis in Low Precision

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

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

In half precision, $u \approx 4.9 \times 10^{-4}$, so $nu = 1$ for $n = 2048$. 
For inner product $x^T y$ of $n$-vectors standard error bound is

$$|fl(x^T y) - x^T y| \leq nu|x|^T|y| + O(u^2).$$

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

What happens when $nu > 1$?
1 Higher Precision

2 Iterative Refinement
Need for Higher Precision

- Long-time simulations.
- Resolving small-scale phenomena.

- Ma and Saunders, *Solving Multiscale Linear Programs Using the Simplex Method in Quadruple Precision*, 2015.
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 $\text{tol}$?
- 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!
Exploit \( A = X \text{diag}(\lambda_i) X^{-1} \Rightarrow f(A) = X \text{diag}(f(\lambda_i)) X^{-1}. \)

**Davies (2007):** “approximate diagonalization”:

```matlab
function F = funm_randomized(A, fun)
d = digits; digits(2*d);
tol = 10^(-d);
E = randn(size(A), class(A));
[V,D] = eig(A + (tol*norm(A,'fro') \* norm(E,'fro'))*E);
F = V*diag(fun(diag(D)))/V;
digits(d)
```

- Perturbation ensures diagonalizable.
- Extra precision overcomes the effect of the perturbation.
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:

How Fast is Quadruple Precision Arithmetic?

Compare

- MATLAB double precision,
- Symbolic Math Toolbox, VPA arithmetic, digits(34),

Ratios of times

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

<table>
<thead>
<tr>
<th>Task</th>
<th>mp/double</th>
<th>vpa/double</th>
<th>vpa/mp</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU, $n = 250$</td>
<td>98</td>
<td>25,000</td>
<td>255</td>
</tr>
<tr>
<td>eig, nonsymm, $n = 125$</td>
<td>75</td>
<td>6,020</td>
<td>81</td>
</tr>
<tr>
<td>eig, symm, $n = 200$</td>
<td>32</td>
<td>11,100</td>
<td>342</td>
</tr>
</tbody>
</table>
Outline

1. Higher Precision

2. 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.)

- Popular up to 1970s, exploiting cheap accumulation of inner products.
Iterative Refinement (1970s, 1980s)

Solve $Ax_0 = b$ by LU factorization.

$\bullet$ $r = b - Ax_0$

$\bullet$ Solve $Ad = r$

$\bullet$ $x_1 = x_0 + d$

*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 $u_f$
- $x_1 = \text{fl}(x_0 + d)$ 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?
Existing Rounding Error Analysis

- **Wilkinson** (1963): fixed-point arithmetic.
- **Moler** (1967): floating-point arithmetic.
- **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 on $f'$err here.
- Allows $\kappa(A)u \gtrsim 1$. 
New Analysis

Assume computed solution to $Ad_i = r_i$ has normwise relative error $O(u_f)$ and satisfies

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

Define $\mu_i$ by

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

and note that

$$\kappa_\infty(A)^{-1} \leq \mu_i \leq 1.$$
\[ |A| = (|a_{ij}|). \]

\[
\text{cond}(A, x) = \frac{\| |A^{-1}| |A||x| \|_\infty}{\|x\|_\infty},
\]

\[
\text{cond}(A) = \text{cond}(A, e) = \| |A^{-1}| |A| \|_\infty,
\]

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

\[ 1 \leq \text{cond}(A, x) \leq \text{cond}(A) \leq \kappa_\infty(A). \]
Convergence Result

Theorem (Carson & H, 2018)

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

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

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

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

Analogous standard bound would have

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

<table>
<thead>
<tr>
<th>$u_f$</th>
<th>$u$</th>
<th>$u_r$</th>
<th>$\kappa_\infty(A)$</th>
<th>Backward error norm</th>
<th>comp</th>
<th>Forward error</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>S</td>
<td>S</td>
<td>$10^4$</td>
<td>S</td>
<td>S</td>
<td>$\text{cond}(A, x) \cdot S$</td>
</tr>
<tr>
<td>H</td>
<td>S</td>
<td>D</td>
<td>$10^4$</td>
<td>S</td>
<td>S</td>
<td>$\text{S}$</td>
</tr>
<tr>
<td>H</td>
<td>D</td>
<td>D</td>
<td>$10^4$</td>
<td>D</td>
<td>D</td>
<td>$\text{cond}(A, x) \cdot D$</td>
</tr>
<tr>
<td>H</td>
<td>D</td>
<td>Q</td>
<td>$10^4$</td>
<td>D</td>
<td>D</td>
<td>$\text{D}$</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>S</td>
<td>$10^8$</td>
<td>S</td>
<td>S</td>
<td>$\text{cond}(A, x) \cdot S$</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>D</td>
<td>$10^8$</td>
<td>S</td>
<td>S</td>
<td>$\text{S}$</td>
</tr>
<tr>
<td>S</td>
<td>D</td>
<td>D</td>
<td>$10^8$</td>
<td>D</td>
<td>D</td>
<td>$\text{cond}(A, x) \cdot D$</td>
</tr>
<tr>
<td>S</td>
<td>D</td>
<td>Q</td>
<td>$10^8$</td>
<td>D</td>
<td>D</td>
<td>$\text{D}$</td>
</tr>
</tbody>
</table>
## Results (2): HSD vs. SSD

<table>
<thead>
<tr>
<th>$u_f$</th>
<th>$u$</th>
<th>$u_r$</th>
<th>$\kappa_\infty(A)$</th>
<th>Backward error norm</th>
<th>comp</th>
<th>Forward error</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>S</td>
<td>S</td>
<td>$10^4$</td>
<td>S</td>
<td>S</td>
<td>$\text{cond}(A, x) \cdot S$</td>
</tr>
<tr>
<td>H</td>
<td>S</td>
<td>D</td>
<td>$10^4$</td>
<td>S</td>
<td>S</td>
<td>$S$</td>
</tr>
<tr>
<td>H</td>
<td>D</td>
<td>D</td>
<td>$10^4$</td>
<td>D</td>
<td>D</td>
<td>$\text{cond}(A, x) \cdot D$</td>
</tr>
<tr>
<td>H</td>
<td>D</td>
<td>Q</td>
<td>$10^4$</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>S</td>
<td>$10^8$</td>
<td>S</td>
<td>S</td>
<td>$\text{cond}(A, x) \cdot S$</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>D</td>
<td>$10^8$</td>
<td>S</td>
<td>S</td>
<td>$S$</td>
</tr>
<tr>
<td>S</td>
<td>D</td>
<td>D</td>
<td>$10^8$</td>
<td>D</td>
<td>D</td>
<td>$\text{cond}(A, x) \cdot D$</td>
</tr>
<tr>
<td>S</td>
<td>D</td>
<td>Q</td>
<td>$10^8$</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>
### Results (2): HSD vs. SSD

<table>
<thead>
<tr>
<th>(u_f)</th>
<th>(u)</th>
<th>(u_r)</th>
<th>(\kappa_\infty(A))</th>
<th>(|u|) norm</th>
<th>(|\Delta u|) comp</th>
<th>(|\Delta x|)</th>
<th>(|\Delta b|)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>S</td>
<td>S</td>
<td>(10^4)</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>H</td>
<td>S</td>
<td>D</td>
<td>(10^4)</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>H</td>
<td>D</td>
<td>D</td>
<td>(10^4)</td>
<td>D</td>
<td>D</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>H</td>
<td>D</td>
<td>Q</td>
<td>(10^4)</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>S</td>
<td>(10^8)</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>D</td>
<td>(10^8)</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>S</td>
<td>D</td>
<td>D</td>
<td>(10^8)</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>S</td>
<td>D</td>
<td>Q</td>
<td>(10^8)</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

Can we get the benefit of “HSD” while allowing a larger range of \(\kappa_\infty(A)\)?
Recall that the convergence condition is

\[ \phi_i = 2u_f \min(\text{cond}(A), \kappa_\infty(A) \mu_i) + u_f \theta_i \ll 1. \]

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

Recall that

\[ \frac{\|d_i - \hat{d}_i\|_\infty}{\|d_i\|} \leq u_f \theta_i, \]

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

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 $\theta_i$

- $u_f \theta_i$ 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 $\hat{L}$ and $\hat{U}$ are computed LU factors of $A$ from GEPP then

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

even for $\kappa(A) \gg u^{-1}$. 
To compute the updates $d_i$ we apply GMRES to

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

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

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

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

Benefits of GMRES-IR

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

<table>
<thead>
<tr>
<th>$u_f$</th>
<th>$u$</th>
<th>$u_r$</th>
<th>$\kappa_\infty(A)$</th>
<th>Backward error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU</td>
<td>H</td>
<td>D</td>
<td>Q</td>
<td>10^4</td>
</tr>
<tr>
<td>LU</td>
<td>S</td>
<td>D</td>
<td>Q</td>
<td>10^8</td>
</tr>
</tbody>
</table>
Benefits of GMRES-IR

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

<table>
<thead>
<tr>
<th></th>
<th>$u_f$</th>
<th>$u$</th>
<th>$u_r$</th>
<th>$\kappa_\infty(A)$</th>
<th>Backward error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>nrm cmp F’error</td>
</tr>
<tr>
<td>LU</td>
<td>H</td>
<td>D</td>
<td>Q</td>
<td>$10^4$</td>
<td>D D D D</td>
</tr>
<tr>
<td>LU</td>
<td>S</td>
<td>D</td>
<td>Q</td>
<td>$10^8$</td>
<td>D D D D</td>
</tr>
<tr>
<td>GMRES-IR</td>
<td>H</td>
<td>D</td>
<td>Q</td>
<td>$10^{12}$</td>
<td>D D D D</td>
</tr>
<tr>
<td>GMRES-IR</td>
<td>S</td>
<td>D</td>
<td>Q</td>
<td>$10^{16}$</td>
<td>D D D D</td>
</tr>
</tbody>
</table>
Benefits of GMRES-IR

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

<table>
<thead>
<tr>
<th></th>
<th>$u_f$</th>
<th>$u$</th>
<th>$u_r$</th>
<th>$\kappa_\infty(A)$</th>
<th>Backward error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU</td>
<td>H</td>
<td>D</td>
<td>Q</td>
<td>$10^4$</td>
<td>D D D D</td>
</tr>
<tr>
<td>LU</td>
<td>S</td>
<td>D</td>
<td>Q</td>
<td>$10^8$</td>
<td>D D D D</td>
</tr>
<tr>
<td>GMRES-IR</td>
<td>H</td>
<td>D</td>
<td>Q</td>
<td>$10^{12}$</td>
<td>D D D D</td>
</tr>
<tr>
<td>GMRES-IR</td>
<td>S</td>
<td>D</td>
<td>Q</td>
<td>$10^{16}$</td>
<td>D D D D</td>
</tr>
</tbody>
</table>

How many GMRES iterations are required?

Some tests with $100 \times 100$ matrices . . .
Test 1: LU-IR, \((u_f, u, u_r) = (S, D, D)\)

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

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

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

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

\[
\kappa_\infty(A) \approx 10^4, \quad \sigma_i = \alpha^i
\]

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

![Graph showing convergence of GMRES-IR with refinement steps and error values.](image)
$\kappa_\infty(A) \approx 10^2$, 1 small $\sigma_i$, GMRES its (8,8,8) Convergence
Test 3: GMRES-IR, \((u_f, u, u_r) = (H, D, Q)\)

\[
\kappa_\infty(A) \approx 10^{12}, \quad \sigma_i = \alpha^i, \text{ GMRES } (100, 100)
\]

Take \(x_0 = 0\) because of overflow!

Convergence
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.

IR with LU in lower precision ⇒ twice as fast as trad. IR, albeit restricted $\kappa(A)$.

Judicious use of a little high precision can bring major benefits.

- **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.
SIAM Announces Class of 2018 Fellows

Society for Industrial and Applied Mathematics recognizes distinguished work through Fellows Program

Society for Industrial and Applied Mathematics (SIAM) is pleased to announce the 2018 Class of SIAM Fellows. These distinguished members were nominated for their exemplary research as well as outstanding service to the community. Through their contributions, SIAM Fellows help advance the fields of applied mathematics and computational science. These individuals will be recognized for their achievements during the SIAM Annual Meeting, happening July 9-13 in Portland, OR.

SIAM congratulates these 28 esteemed members of the community, listed below in alphabetical order:

Alex Pothen, Purdue University, is being recognized for advances in combinatorial algorithms for scientific applications, and leadership in founding the combinatorial scientific computing community.
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.

G. Beliakov and Y. Matiyasevich.
A parallel algorithm for calculation of determinants and minors using arbitrary precision arithmetic.
E. Carson and N. J. Higham. 
A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. 

E. Carson and N. J. Higham. 
Accelerating the solution of linear systems by iterative refinement in three precisions. 

M. Courbariaux, Y. Bengio, and J.-P. David. 
*ArXiv* preprint 1412.7024v5.
E. B. Davies.
Approximate diagonalization.

N. J. Dingle and N. J. Higham.
Reducing the influence of tiny normwise relative errors on performance profiles.

N. J. Higham.
Iterative refinement for linear systems and LAPACK.

Y. Kobayashi and T. Ogita.  
A fast and efficient algorithm for solving ill-conditioned linear systems.  

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).  
More flops or more precision? Accuracy parameterizable linear equation solvers for model predictive control.

K. Scheinberg.
Evolution of randomness in optimization methods for supervised machine learning.