MANCHESTER 1824

The University of Manchester

M\crNA
Manchester Numerical Analysis

Multiprecision Inverse Computation

Michael Connolly
School of Mathematics
The University of Manchester

Numerical Linear Algebra Group Meeting
3 April 2019

Newton-Schulz Method

For $A \in \mathbb{R}^{m \times n}$ we have the following iterative scheme for computing A^+:

- $X_{k+1} = (2I - X_k A)X_k$, $X_0 = \alpha A^T$.
- Denoting $E_k = I - X_k A$, we have that $E_{k+1} = E_k^2 = \cdots = E_0^{2^{k+1}}$ and so require $\|I - \alpha A^T A\|_2 < 1$ for convergence.
- Require $0 < \alpha < 2/\|A\|_2^2$.
- For all results shown we use $\alpha = 1/(\|A\|_1 \|A\|_\infty)$.

Newton-Schulz Method

Convergence test: Define

$$\text{res}_L = \frac{\|I - X_k A\|_\infty}{\|X_k\|_\infty \|A\|_\infty}.$$

We assume convergence when $\text{res}_L < \epsilon$, the machine epsilon of the precision in which we are working.

Newton-Schulz Method

Convergence test: Define

$$\text{res}_L = \frac{\|I - X_k A\|_\infty}{\|X_k\|_\infty \|A\|_\infty}.$$

We assume convergence when $\text{res}_L < \epsilon$, the machine epsilon of the precision in which we are working.

- Total iteration count bounded below by $2\log_2(\kappa_2(A))$.
- If $\kappa_2(A) = 10^8$, we require at least 53 iterations.

Newton-Schulz Method

Convergence test: Define

$$\text{res}_L = \frac{\|I - X_k A\|_\infty}{\|X_k\|_\infty \|A\|_\infty}.$$

We assume convergence when $\text{res}_L < \epsilon$, the machine epsilon of the precision in which we are working.

- Total iteration count bounded below by $2\log_2(\kappa_2(A))$.
- If $\kappa_2(A) = 10^8$, we require at least 53 iterations.

Expensive!

Increasing Precision

- Given a sequence of increasing precisions, we perform the iteration in a given precision until we obtain convergence and then increase the precision.
- That is $\text{res}_L < \epsilon_p$, with ϵ_p the machine epsilon of the relevant precision.
- For all experiments shown, we use **fp16** $\rightarrow$ **fp32** $\rightarrow$ **fp64**.

Simulating fp16

- We use the newly released `chop` from Higham & Pranesh (2019) to simulate IEEE half precision in MATLAB.
- **`https://github.com/higham/chop`**
- Perform each scalar operation in **fp64** or **fp32** and round to **fp16**.

Computing $AB = \sum_{k=1}^{n} A(:,k)B(k,:)$, we call `chop` only $\mathcal{O}(n)$ times.

Numerical Results

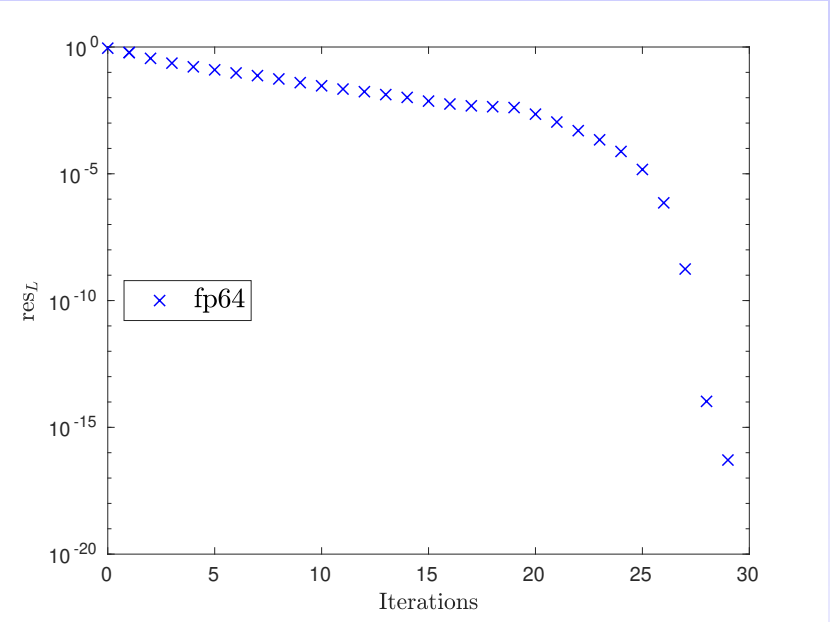


Figure 1: **fp64** implementation.

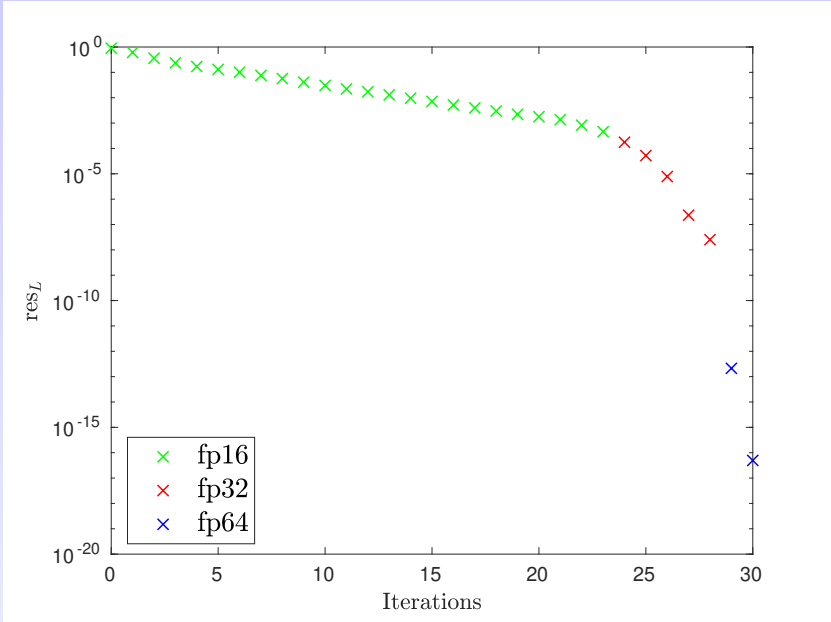


Figure 2: Increasing precision implementation.

In both we have A sampled from $\mathcal{N}(0,1)$ and $n = 100$. Here $\kappa_2(A) \approx 7e+02$.

Numerical Results

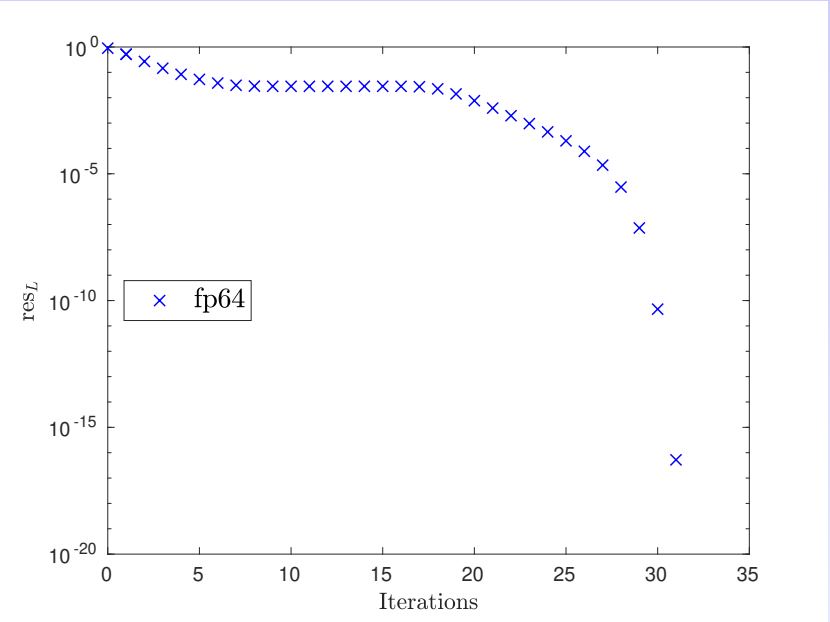


Figure 3: **fp64** implementation.

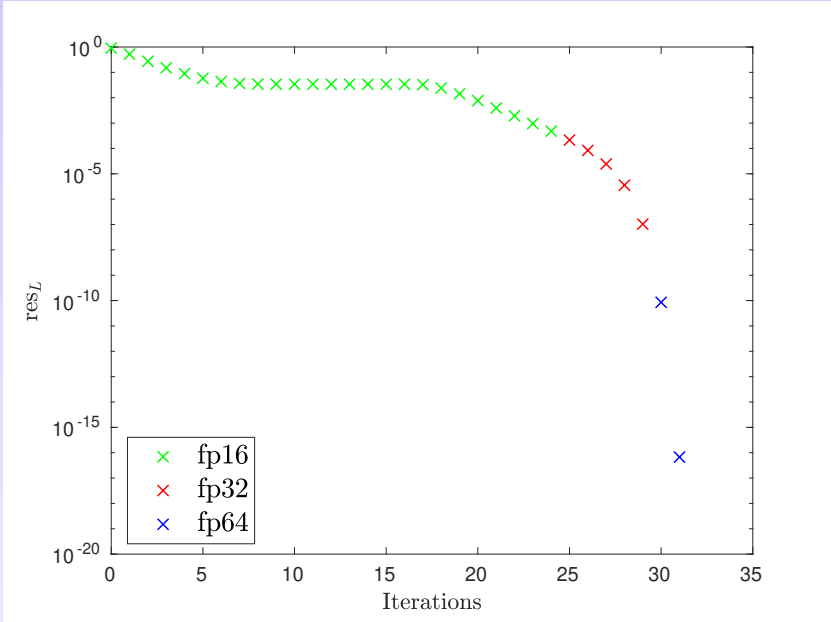


Figure 4: Increasing precision implementation.

Here A is from `gallery('randsvd')` with $n = 100$ and $\kappa_2(A) = 1e+03$.

Acceleration

Pan & Schreiber (1991) introduce two forms of acceleration.

- **Scaling:** Compute α at each iteration so that we now have $X_{k+1} = \alpha_{k+1}(2I - X_k A)X_k$.
- **Cubic polynomials:** For $\rho > 0$ and $T_k = X_k A$, compute $X_{k+1} = (1/\rho)(T_k^2 - (2+\rho)T_k + (1+2\rho)I)X_k$.

Acceleration

Pan & Schreiber (1991) introduce two forms of acceleration.

- **Scaling:** Compute α at each iteration so that we now have $X_{k+1} = \alpha_{k+1}(2I - X_k A)X_k$.
- **Cubic polynomials:** For $\rho > 0$ and $T_k = X_k A$, compute $X_{k+1} = (1/\rho)(T_k^2 - (2+\rho)T_k + (1+2\rho)I)X_k$.

We compute $\delta = \|T_k - T_k^2\|_F$ and $\bar{\delta} = \delta/n$. If $\text{trace}(T_k) < n - 1/2$ then:

- If $\delta > 1/4$ and $\bar{\delta} < 1/4$, we accelerate by scaling.
- If $\delta < 1/4$ we accelerate with cubic polynomials.
- Here $\rho = 1/2 - \sqrt{1/4 - \delta}$, $\bar{\rho} = 1/2 - \sqrt{1/4 - \bar{\delta}}$ and $\alpha_{k+1} = 2/(1 + (2 - \bar{\rho})\bar{\rho})$.

Preconditioning the iteration

Computing LU factors in low precision, we can then compute $\tilde{A} = U^{-1}L^{-1}A$.

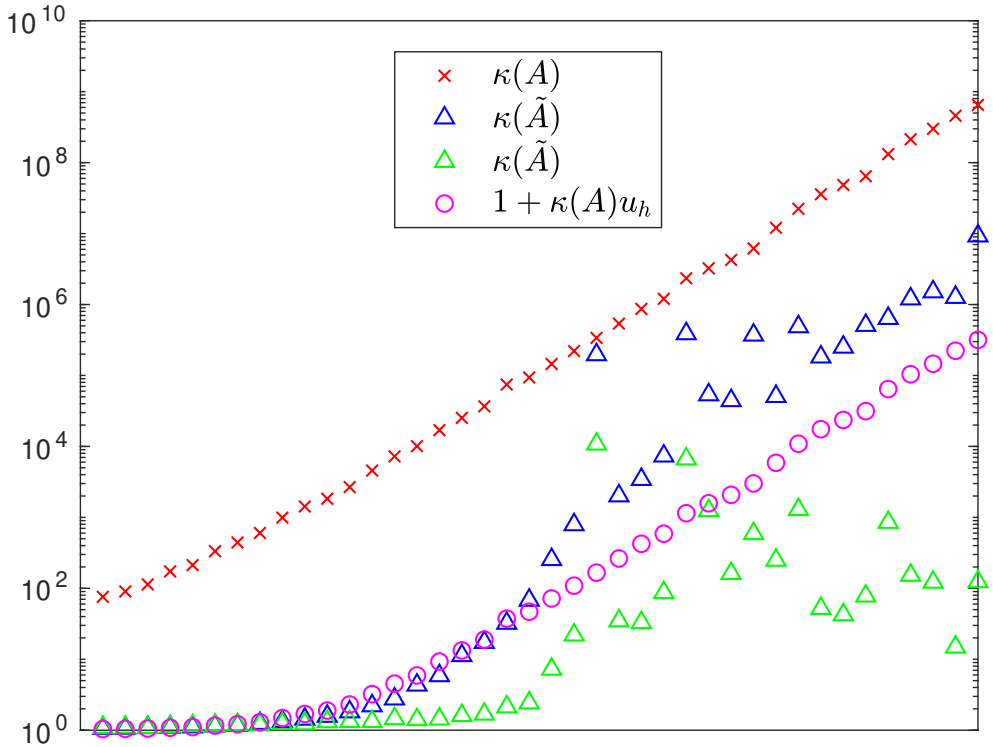
We don't form these inverses explicitly, but instead compute $\tilde{A}$ by a set of triangular solves.

Higham & Carson (2017) verify the relation

$$\kappa_\infty(\tilde{A}) \approx 1 + \kappa_\infty(A)u.$$

Preconditioning the Iteration

- Must compute the solves in **fp64** to recover a solution accurate to **fp64**.
- How does this mixture of precisions affect the approximation?



Improved Iteration

LU factors computed in **fp16**, triangular solves computed in **fp64**.

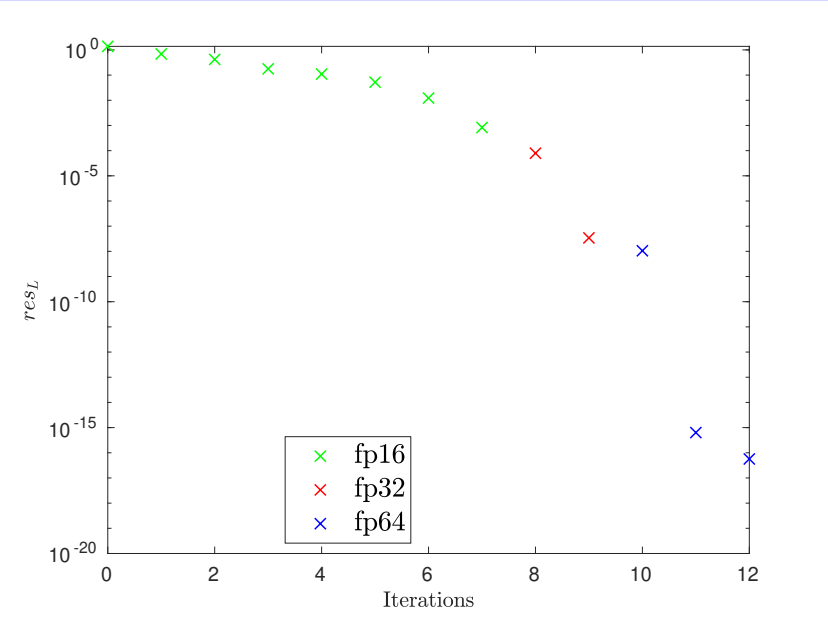


Figure 6: A sampled from $\mathcal{N}(0,1)$ with $\kappa_2(A) \approx 1e+03$.

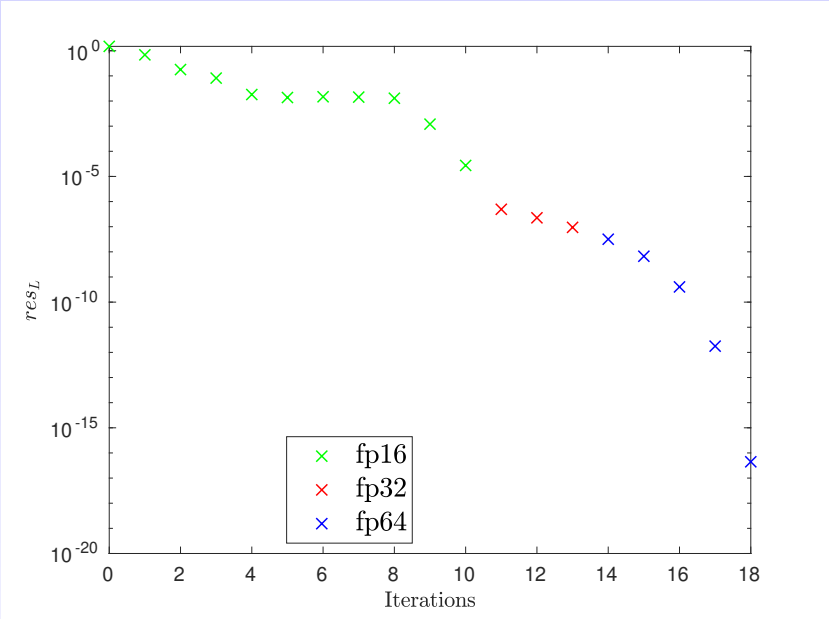


Figure 7: $\kappa_2(A) = 1e+08$ with one small singular value.

Iterative Refinement

Higham & Carson (2017, 2018) present results from iterative refinement in three precisions (u_f, u, u_r).

Algorithm 1 Iterative refinement in three precisions.

1: Compute LU factors $A = \hat{L}\hat{U}$ in precision u_f.
2: Solve $Ax_0 = b$ in precision u_f using the LU factors and store at precision u.
3: **for** $i = 0 : \infty$ **do**
4: Compute $r_i = b - Ax_i$ in precision u_r
5: and round to precision u.
6: Solve $\tilde{A}d_i = \hat{U}^{-1}\hat{L}^{-1}r_i$ at precision u by GMRES.
7: $x_{i+1} = x_i + d_i$ at precision u.
8: **end for**

Alternative Preconditioning

- Instead of the LU factors, we precondition the GMRES solves with B, a low precision approximation to A^{-1} obtained iteratively.

Alternative Preconditioning

- Instead of the LU factors, we precondition the GMRES solves with B, a low precision approximation to A^{-1} obtained iteratively.
- We compare results of these variants of iterative refinement:

	$\kappa_2(A)$	$\kappa_\infty(A)$	LU cost	B cost
1	$1e+04$	$9e+04$	$2(17)$	$2(4)$
2	$1e+06$	$7.4e+06$	$3(185)$	$2(75)$
3	$1e+09$	$2.01e+10$	$-(-)$	$9(45)$

- We show total number of iterative refinement steps and GMRES iterations, shown in parentheses, required.
- 1 and 2 above are the default of `gallery('randsvd')` and 3 contains one small singular value.

Alternative Preconditioning

- Obtaining B much more expensive than LU factors.
- As seen earlier, we need to precondition the Newton-Schulz iteration to obtain convergence. We must also compute relevant triangular solves, done in **fp32** here.
- Try more values of (u_f, u, u_r).

Questions?

References I

V. Pan and R. Schreiber
An improved Newton iteration for the Generalized Inverse of a Matrix, with Applications.
12(5):1109-1130, SIAM J. Sci. Stat. Comput., 1991.

N. J. Higham and S. Pranesh
Simulating low precision floating point arithmetic.
MIMS EPrint 2019.4, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, 2019.

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 II


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.