MAX PLANCK INSTITUTE
FOR DYNAMICS OF COMPLEX
TECHNICAL SYSTEMS
MAGDEBURG

COMPUTATIONAL METHODS IN
SYSTEMS AND CONTROL THEORY

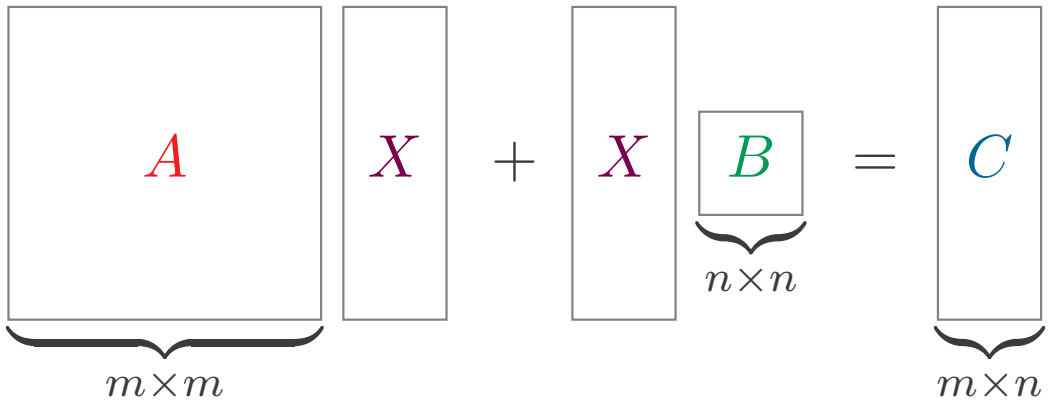
Mixed-precision algorithms for solving the Sylvester matrix equations

Xiaobo Liu
MPI for Dynamics of Complex Technical Systems, Germany

ENUMATH 2025,
Heidelberg University, September 3, 2025

Joint work with A. Dmytryshyn • M. Fasi • N. J. Higham

The Sylvester matrix equations



App. in block diagonalization • perturbation theory for matrix functions • system balancing • control • model reduction, etc.

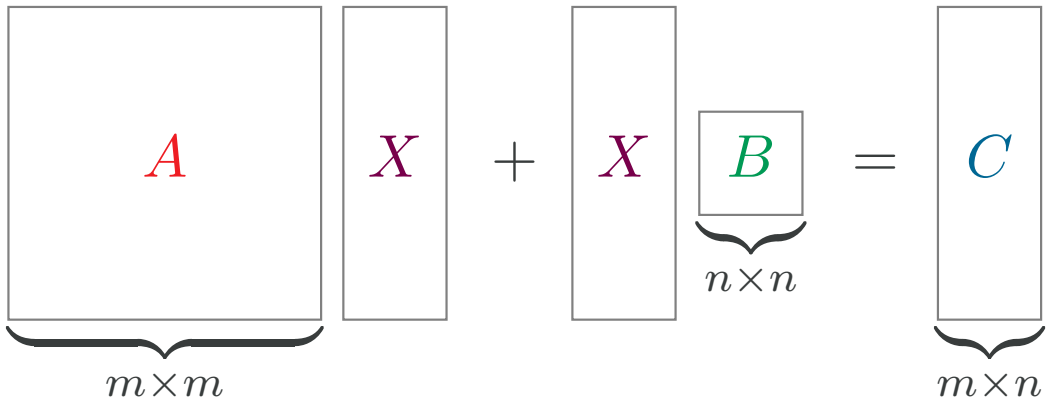
The setting:

- Dense coefficients of moderate size $\rightsquigarrow$ say, $\lesssim \mathcal{O}(10^4)$
- No enforced structure

$\rightsquigarrow$ Method of choice: Bartels–Stewart algorithm (**Bartels & Stewart, '72**).

To do: find X utilizing two precisions: high (working) prec. + low prec.

The Sylvester matrix equations



App. in block diagonalization • perturbation theory for matrix functions • system balancing • control • model reduction, etc.

The setting:

- Dense coefficients of moderate size $\rightsquigarrow$ say, $\lesssim \mathcal{O}(10^4)$
- No enforced structure

$\rightsquigarrow$ Method of choice: Bartels–Stewart algorithm (**Bartels & Stewart, '72**).

To do: find X utilizing two precisions: high (working) prec. + low prec.

Floating-point arithmetics

Computers can represent numbers as

$$x = \pm m \cdot 2^e$$

where

- m is a t-digit number in $[0, 2)$
- e is an integer between $e_{\min}$ and $e_{\max}$ (in IEEE 754 $e_{\min} = 1 - e_{\max}$)

Table: Parameters of five floating point formats.

Type	Signif. bits (t)	Exp. bits	Range	$u = 2^{-t}$
bfloat16	8	8	$10^{\pm 38}$	3.9×10^{-3}
binary16	11	5	$10^{\pm 5}$	4.9×10^{-4}
TensorFloat-32	11	8	$10^{\pm 38}$	4.9×10^{-4}
binary32	24	8	$10^{\pm 38}$	6.0×10^{-8}
binary64	53	11	$10^{\pm 308}$	1.1×10^{-16}

- Lower precision $\rightsquigarrow$ faster flops, less comm., lower energy consumption.

Floating-point arithmetics

Computers can represent numbers as

$$x = \pm m \cdot 2^e$$

where

- m is a t-digit number in $[0, 2)$
- e is an integer between $e_{\min}$ and $e_{\max}$ (in IEEE 754 $e_{\min} = 1 - e_{\max}$)

Table: Parameters of five floating point formats.

Type	Signif. bits (t)	Exp. bits	Range	$u = 2^{-t}$
bfloat16	8	8	$10^{\pm 38}$	3.9×10^{-3}
binary16	11	5	$10^{\pm 5}$	4.9×10^{-4}
TensorFloat-32	11	8	$10^{\pm 38}$	4.9×10^{-4}
binary32	24	8	$10^{\pm 38}$	6.0×10^{-8}
binary64	53	11	$10^{\pm 308}$	1.1×10^{-16}

- Lower precision $\rightsquigarrow$ faster flops, less comm., lower energy consumption.

Floating-point arithmetics

Computers can represent numbers as

$$x = \pm m \cdot 2^e$$

where

- m is a t-digit number in $[0, 2)$
- e is an integer between $e_{\min}$ and $e_{\max}$ (in IEEE 754 $e_{\min} = 1 - e_{\max}$)

Table: Parameters of five floating point formats.

Type	Signif. bits (t)	Exp. bits	Range	$u = 2^{-t}$
bfloat16	8	8	$10^{\pm 38}$	3.9×10^{-3}
binary16	11	5	$10^{\pm 5}$	4.9×10^{-4}
TensorFloat-32	11	8	$10^{\pm 38}$	4.9×10^{-4}
binary32	24	8	$10^{\pm 38}$	6.0×10^{-8}
binary64	53	11	$10^{\pm 308}$	1.1×10^{-16}

- Lower precision $\rightsquigarrow$ faster flops, less comm., lower energy consumption.

Use of low and mixed precisions

When to use low precision?

- Low accuracy is needed
- Other sources of errors:
  model reduction, approximation (e.g., low rank, discretization)
- Self-correction mechanism (e.g., iterative refinement) ✓

$\rightsquigarrow$ Mixed-precision algorithm selects different precisions for different parts of computation to improve performance while guarantee accuracy and stability.

Use of low and mixed precisions

When to use low precision?

- Low accuracy is needed
- Other sources of errors:
  model reduction, approximation (e.g., low rank, discretization)
- Self-correction mechanism (e.g., iterative refinement) ✓

$\rightsquigarrow$ Mixed-precision algorithm selects different precisions for different parts of computation to improve performance while guarantee accuracy and stability.

Use of low and mixed precisions

When to use low precision?

- Low accuracy is needed
- Other sources of errors:
  model reduction, approximation (e.g., low rank, discretization)
- Self-correction mechanism (e.g., iterative refinement) ✓

$\rightsquigarrow$ Mixed-precision algorithm selects different precisions for different parts of computation to improve performance while guarantee accuracy and stability.

The Bartels–Stewart algorithm

$$A\,X + X\,B = C$$

1. Comput. Schur decompositions:

$$A = U_A\,T_A\,U_A^*, \qquad B = U_B\,T_B\,U_B^*$$

2. Solve by substitution

$$T_A\,Y + Y\,T_B = \widetilde{C}, \quad \widetilde{C} = U_A^* C U_B$$

3. Return $X := U_A Y U_B^*$

The Bartels–Stewart algorithm

$$A\,X + X\,B = C$$

1. Comput. Schur decompositions:

$$A = U_A\,T_A\,U_A^*, \qquad B = U_B\,T_B\,U_B^*$$

2. Solve by substitution

$$T_A\,Y + Y\,T_B = \widetilde{C}, \qquad \widetilde{C} = U_A^* C U_B$$

3. Return $X := U_A Y U_B^*$

The Bartels–Stewart algorithm

$$A\,X + X\,B = C$$

1. Comput. Schur decompositions:

$$A = U_A\,T_A\,U_A^*, \qquad B = U_B\,T_B\,U_B^*$$

2. Solve by substitution

$$T_A\,Y + Y\,T_B = \widetilde{C}, \qquad \widetilde{C} = U_A^* C U_B$$

3. Return $X := U_A Y U_B^*$

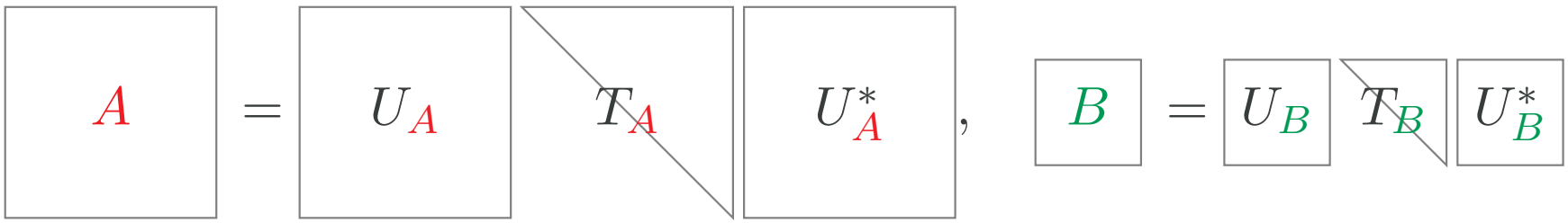
The Bartels–Stewart algorithm

$$A\,X + X\,B = C$$

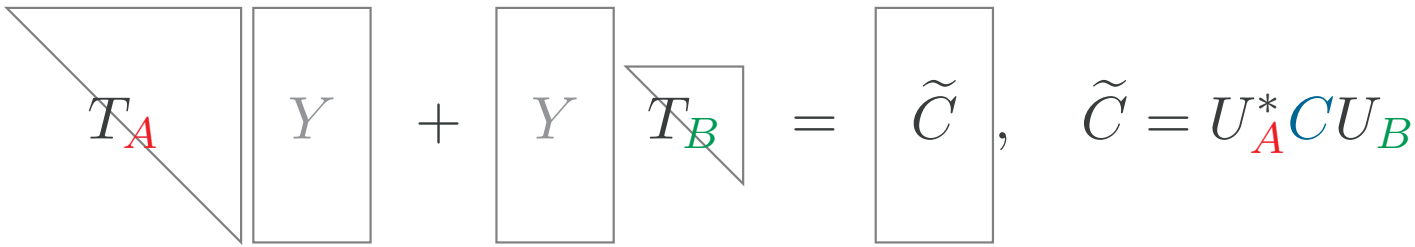
$$\Updownarrow$$

$$U_A^* U_A T_A U_A^* X U_B + U_A^* X U_B T_B U_B^* U_B = U_A^* C U_B$$

1. Comput. Schur decompositions:



2. Solve by substitution



3. Return $X := U_A Y\, U_B^*$

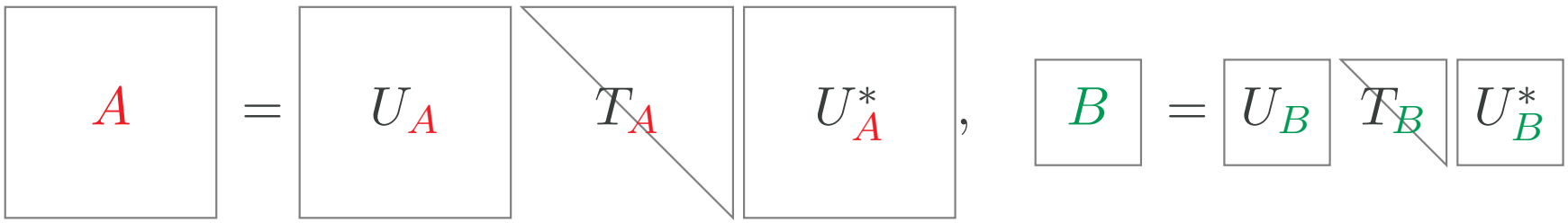
The Bartels–Stewart algorithm

$$A\,X + X\,B = C$$

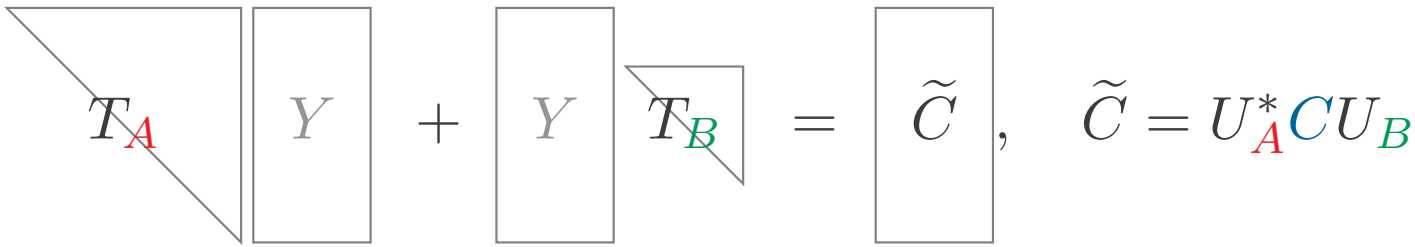
$$\Updownarrow$$

$$U_A^* U_A T_A U_A^* X U_B + U_A^* X U_B T_B U_B^* U_B = U_A^* C U_B$$

1. Comput. Schur decompositions:



2. Solve by substitution



3. Return $X := U_A Y U_B^*$

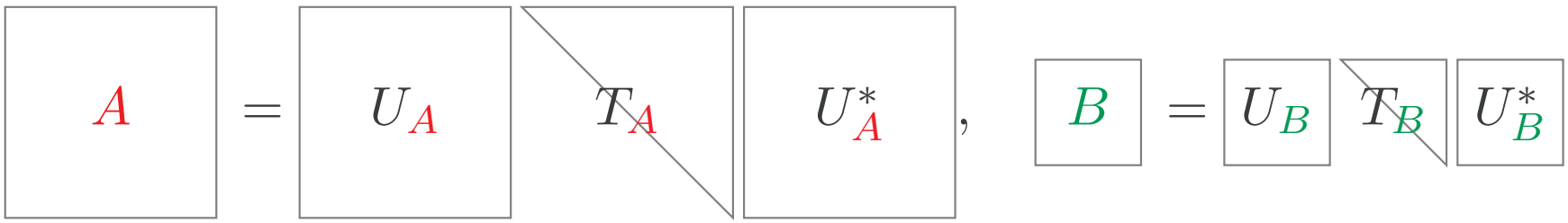
The Bartels–Stewart algorithm

$$A\,X + X\,B = C$$

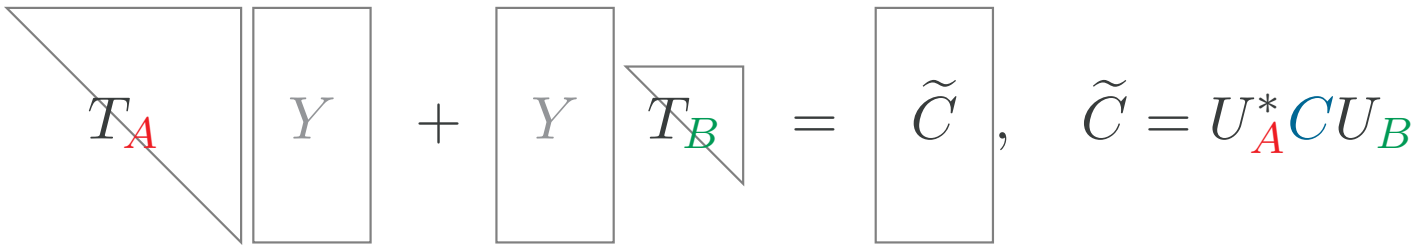
$$\Updownarrow$$

$$U_A^* U_A T_A U_A^* X U_B + U_A^* X U_B T_B U_B^* U_B = U_A^* C U_B$$

1. Comput. Schur decompositions:



2. Solve by substitution



3. Return $X := U_A Y U_B^*$

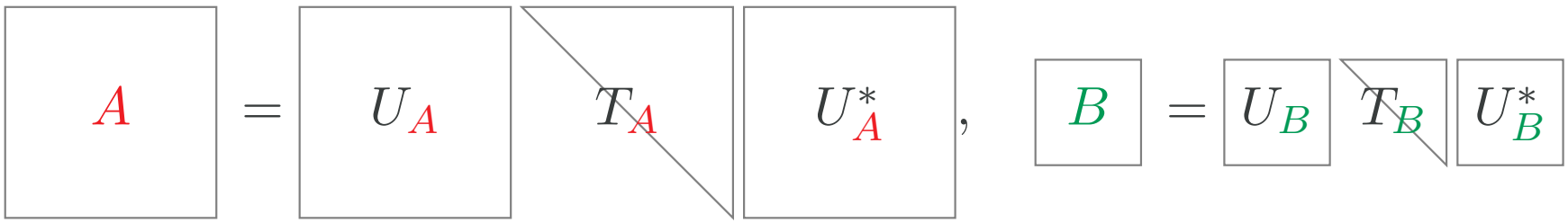
The Bartels–Stewart algorithm

$$A\,X + X\,B = C$$

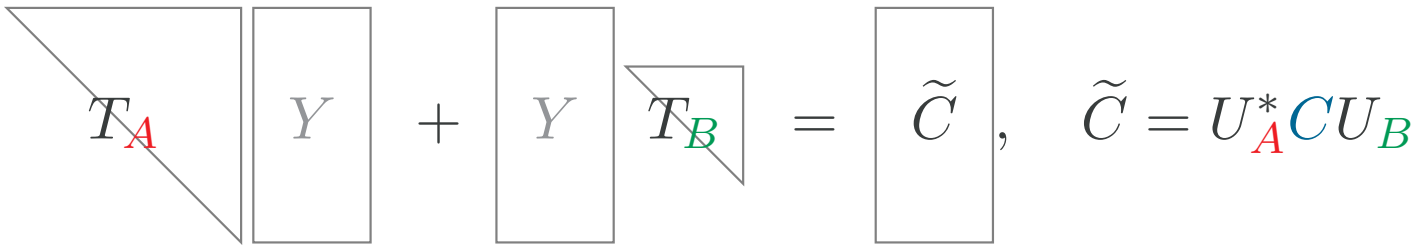
$$\Updownarrow$$

$$T_A\,Y + Y\,T_B = U_A^* C U_B, \quad Y = U_A^* X U_B$$

1. Comput. Schur decompositions:



2. Solve by substitution



3. Return $X := U_A Y\, U_B^*$

Towards a mixed-precision algorithm

1. Compute the Schur decomposition $A =: U_A T_A U_A^*$ ▷ $25m^3$
2. Compute the Schur decomposition $B =: U_B T_B U_B^*$ ▷ $25n^3$
3. Compute $\widetilde{C} := U_A^* C\, U_B$ ▷ $2mn(m+n)$
4. Solve $T_A\, Y + Y\, T_B = \widetilde{C}$ for Y ▷ $mn(m+n)$
5. Return $X := U_A Y\, U_B^*$ ▷ $2mn(m+n)$

Observations

- Steps 1 and 2 to be performed in low precision
- Low-precision Schur factors $\rightsquigarrow$ steps 3–5 will have to change

$\rightsquigarrow$ Our idea

1. Low-precision triangular factors available $\rightsquigarrow$ Use iterative refinement to get high-precision solution to the quasi-triangular equation
2. Re-orthonormalize or explicitly invert low-precision unitary factors

Towards a mixed-precision algorithm

1. Compute the Schur decomposition $A =: U_A T_A U_A^*$ ▷ $25m^3$
2. Compute the Schur decomposition $B =: U_B T_B U_B^*$ ▷ $25n^3$
3. Compute $\widetilde{C} := U_A^* C U_B$ ▷ $2mn(m+n)$
4. Solve $T_A Y + Y T_B = \widetilde{C}$ for Y ▷ $mn(m+n)$
5. Return $X := U_A Y U_B^*$ ▷ $2mn(m+n)$

Observations

- Steps 1 and 2 to be performed in low precision
- Low-precision Schur factors $\rightsquigarrow$ steps 3–5 will have to change

$\rightsquigarrow$ Our idea

1. Low-precision triangular factors available $\rightsquigarrow$ Use iterative refinement to get high-precision solution to the quasi-triangular equation
2. Re-orthonormalize or explicitly invert low-precision unitary factors

Towards a mixed-precision algorithm

1. Compute the Schur decomposition $A =: U_A T_A U_A^*$ ▷ $25m^3$
2. Compute the Schur decomposition $B =: U_B T_B U_B^*$ ▷ $25n^3$
3. Compute $\widetilde{C} := U_A^* C U_B$ ▷ $2mn(m+n)$
4. Solve $T_A Y + Y T_B = \widetilde{C}$ for Y ▷ $mn(m+n)$
5. Return $X := U_A Y U_B^*$ ▷ $2mn(m+n)$

Observations

- Steps 1 and 2 to be performed in low precision
- Low-precision Schur factors $\rightsquigarrow$ steps 3–5 will have to change

$\rightsquigarrow$ Our idea

1. Low-precision triangular factors available $\rightsquigarrow$ Use iterative refinement to get high-precision solution to the quasi-triangular equation
2. Re-orthonormalize or explicitly invert low-precision unitary factors

Iterative refinement for quasi-triangular equation

Goal: solve $\underbrace{(T_A + \Delta_A)}_{\text{high-prec. } T_A} Y + Y \underbrace{(T_B + \Delta_B)}_{\text{high-prec. } T_B} = \underbrace{U_A^* C U_B}_{\text{unitary to high-prec.}} =: D$

1 $k \leftarrow 0$
2 **while** $\|\Delta_Y\| > \varepsilon$ **do**
3 $\quad R_k \leftarrow D - (T_A + \Delta_A) Y_k + Y_k (T_B + \Delta_B)$
4 $\quad$ Solve $T_A \Delta_Y + \Delta_Y T_B = R_k$ for Δ_Y using Bartels–Stewart
5 $\quad Y_{k+1} \leftarrow Y_k + \Delta_Y$
6 $\quad k \leftarrow k + 1$
7 **end**
8 **return** Y_k

Cost: $3kmn(m+n)$ flops

- Sufficient convergence condition: $\|\Delta_A\|_2 + \|\Delta_B\|_2 < \operatorname{sep}_F(T_A, -T_B)$

Iterative refinement for quasi-triangular equation

Goal: solve $\underbrace{(T_A + \Delta_A)}_{\text{high-prec. } T_A} Y + Y \underbrace{(T_B + \Delta_B)}_{\text{high-prec. } T_B} = \underbrace{U_A^* C U_B}_{\text{unitary to high-prec.}} =: D$

---

1 $k \leftarrow 0$
2 **while** $\|\Delta_Y\| > \varepsilon$ **do**
3 $\quad R_k \leftarrow D - (T_A + \Delta_A) Y_k + Y_k (T_B + \Delta_B)$
4 $\quad$ Solve $T_A \Delta_Y + \Delta_Y T_B = R_k$ for Δ_Y using Bartels–Stewart
5 $\quad Y_{k+1} \leftarrow Y_k + \Delta_Y$
6 $\quad k \leftarrow k + 1$
7 **end**
8 **return** Y_k

Cost: $3kmn(m+n)$ flops

- Sufficient convergence condition: $\|\Delta_A\|_2 + \|\Delta_B\|_2 < \operatorname{sep}_F(T_A, -T_B)$

Iterative refinement for quasi-triangular equation

Goal: solve $\underbrace{(T_A + \Delta_A)}_{\text{high-prec. } T_A} Y + Y \underbrace{(T_B + \Delta_B)}_{\text{high-prec. } T_B} = \underbrace{U_A^* C U_B}_{\text{unitary to high-prec.}} =: D$

---

1 $k \leftarrow 0$
2 **while** $\|\Delta_Y\| > \varepsilon$ **do**
3 $\quad R_k \leftarrow D - (T_A + \Delta_A) Y_k + Y_k (T_B + \Delta_B)$
4 $\quad$ Solve $T_A \Delta_Y + \Delta_Y T_B = R_k$ for Δ_Y using Bartels–Stewart
5 $\quad Y_{k+1} \leftarrow Y_k + \Delta_Y$
6 $\quad k \leftarrow k + 1$
7 **end**
8 **return** Y_k

Cost: $3kmn(m+n)$ flops

- Sufficient convergence condition: $\|\Delta_A\|_2 + \|\Delta_B\|_2 < \operatorname{sep}_F(T_A, -T_B)$

Mixed-precision algorithm

Via re-ortho. low-precision unitary factors

1 Compute Schur decompositions $A = U_A T_A U_A^*$ ▷ in low-precision u_ℓ

2 Compute Schur decompositions $B = U_B T_B U_B^*$

3 Compute QR decompositions $U_A = Q_A R_A$, $U_B = Q_B R_B$ ▷ re-ortho.

4 $\Delta_A \leftarrow Q_A^* A Q_A - T_A$

5 $\Delta_B \leftarrow Q_B^* B Q_B - T_B$

6 $D \leftarrow Q_A^* C Q_B$

7 Solve $T_A Y + Y T_B = D$ for Y

8 **while** $\|\Delta_Y\| > \varepsilon$ **do**

9 $\quad R \leftarrow D - (T_A + \Delta_A) Y + Y (T_B + \Delta_B)$

10 $\quad$ Solve $T_A \Delta_Y + \Delta_Y T_B = R$ for Δ_Y using Bartels–Stewart

11 $\quad Y \leftarrow Y + \Delta_Y$

12 **end**

13 $X \leftarrow Q_A Y Q_B^*$

Mixed-precision algorithm

Via inverting low-precision unitary factors

1 Compute Schur decompositions $A = U_A T_A U_A^*$ ▷ in low-precision u_ℓ

2 Compute Schur decompositions $B = U_B T_B U_B^*$

3 Compute LU decompositions of U_A^* and U_B

4 $\Delta_A \leftarrow U_A^* A U_A^{-*} - T_A$

5 $\Delta_B \leftarrow U_B^{-1} B U_B - T_B$

6 $D \leftarrow U_A^* C U_B$

7 Solve $T_A Y + Y T_B = D$ for Y

8 **while** $\|\Delta_Y\| > \varepsilon$ **do**

9 $\quad R \leftarrow D - (T_A + \Delta_A) Y + Y (T_B + \Delta_B)$

10 $\quad$ Solve $T_A \Delta_Y + \Delta_Y T_B = R$ for Δ_Y using Bartels–Stewart

11 $\quad Y \leftarrow Y + \Delta_Y$

12 **end**

13 $X \leftarrow U_A^{-*} Y U_B^{-1}$

When is low precision worth it?

Cost of mixed precision algorithms

- Re-orthonornalization
  - $25(m^3 + n^3) + mn(m + n)$ low-precision flops
  - $6(m^3 + n^3) + (4 + 3k)mn(m + n)$ high-precision flops
- Inversion
  - $25(m^3 + n^3) + mn(m + n)$ low-precision flops
  - $4\frac{2}{3}(m^3 + n^3) + (4 + 3k)mn(m + n)$ high-precision flops

Cost of Bartels–Stewart

- $25(m^3 + n^3) + 5mn(m + n)$ high-precision flops

How do these costs compare?

When is low precision worth it?

Cost of mixed precision algorithms

- Re-orthonornalization
  - $25(m^3 + n^3) + mn(m + n)$ low-precision flops
  - $6(m^3 + n^3) + (4 + 3k)mn(m + n)$ high-precision flops
- Inversion
  - $25(m^3 + n^3) + mn(m + n)$ low-precision flops
  - $4\frac{2}{3}(m^3 + n^3) + (4 + 3k)mn(m + n)$ high-precision flops

Cost of Bartels–Stewart

- $25(m^3 + n^3) + 5mn(m + n)$ high-precision flops

How do these costs compare?

Computational cost in mixed precision

Model for computational cost

$$\rho = \frac{\text{average time of low-precision flop}}{\text{average time of high-precision flop}}$$

$\rho = 0$ low-precision flops are negligible

$\rho = 1$ low- and high-precision flops have same cost

$\rho > 1$ simulated low precision

$\boxed{0 \leq \rho \ll 1}$ on emerging hardware

Computational cost in mixed precision

Model for computational cost

$$\rho = \frac{\text{average time of low-precision flop}}{\text{average time of high-precision flop}}$$

$\rho = 0$ low-precision flops are negligible

$\rho = 1$ low- and high-precision flops have same cost

$\rho > 1$ simulated low precision

$0 \leq \rho \ll 1$ on emerging hardware

Computational cost in mixed precision

Model for computational cost

$$\rho = \frac{\text{average time of low-precision flop}}{\text{average time of high-precision flop}}$$

$\rho = 0$ low-precision flops are negligible

$\rho = 1$ low- and high-precision flops have same cost

$\rho > 1$ simulated low precision

$\boxed{0 \leq \rho \ll 1}$ on emerging hardware

Computational cost in mixed precision

Normalised computational cost (in high precision)

$$C_N = C_h + \rho \cdot C_\ell$$

- C_h: number of high-precision flops
- C_ℓ: number of low-precision flops

Each flop in C_ℓ is multiplied by ρ.

How many iterations can we afford?

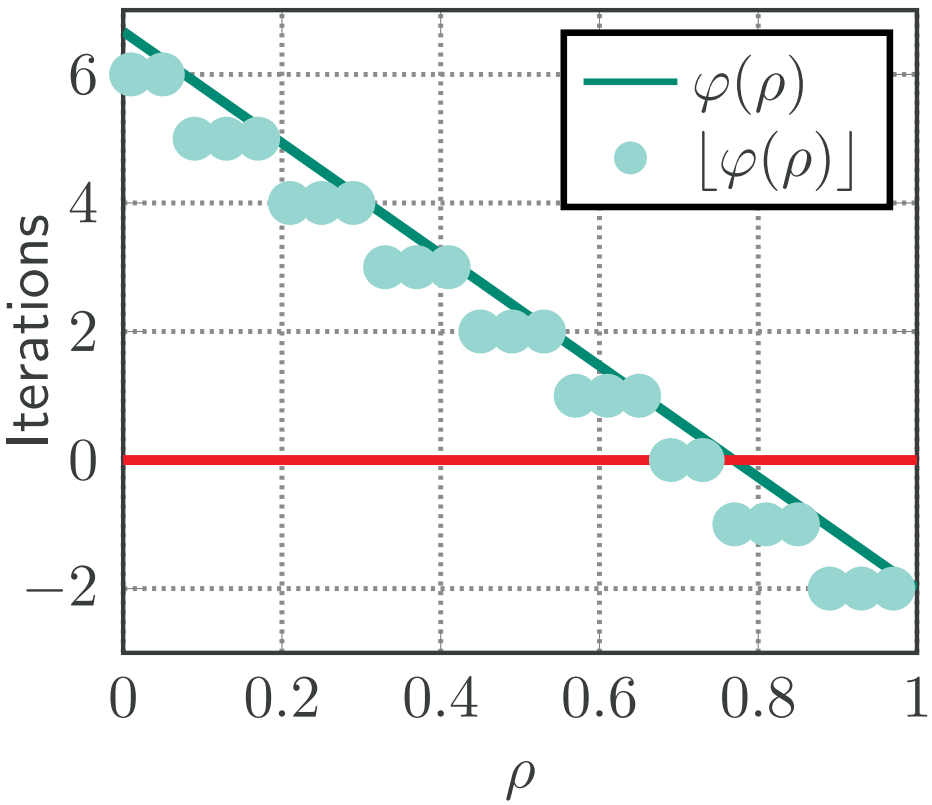
Maximum number of iterations

$$\lfloor \varphi(\rho) \rfloor, \qquad \varphi(\rho) = \frac{C_{BS}(m,n) - C_N^{ni}(m,n)}{C_N^{it}(m,n)}$$

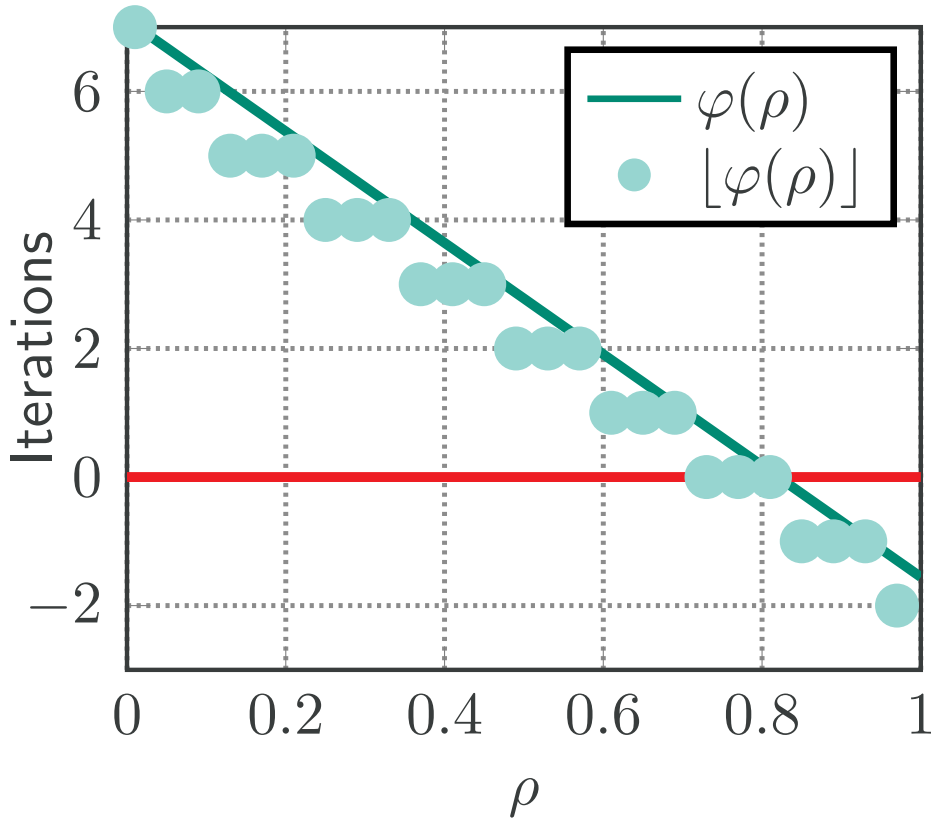
where

- $C_{BS}(m,n)$: cost of Bartels–Stewart in high precision
- $C_N^{ni}(m,n)$: normalised cost of the non-iterative part of new algorithm
- $C_N^{it}(m,n)$: normalised cost of one iteration of new algorithm

At most 6 iterations when $m = n$



(a) Re-orthonormalization-based.



(b) Inversion-based.

- $\rho =$ avg. time of low-precision flop $/$ avg. time of high-precision flop

Experiments setup

Algorithms

- `sylv`: the built-in MATLAB function `sylvester`
- `mp_orth`: MATLAB implementation of re-ortho.-based algorithm
- `mp_inv`: MATLAB implementation of inversion-based algorithm

Refinement tolerance: 1e-12$\cdot \max(m, n)$

Test set

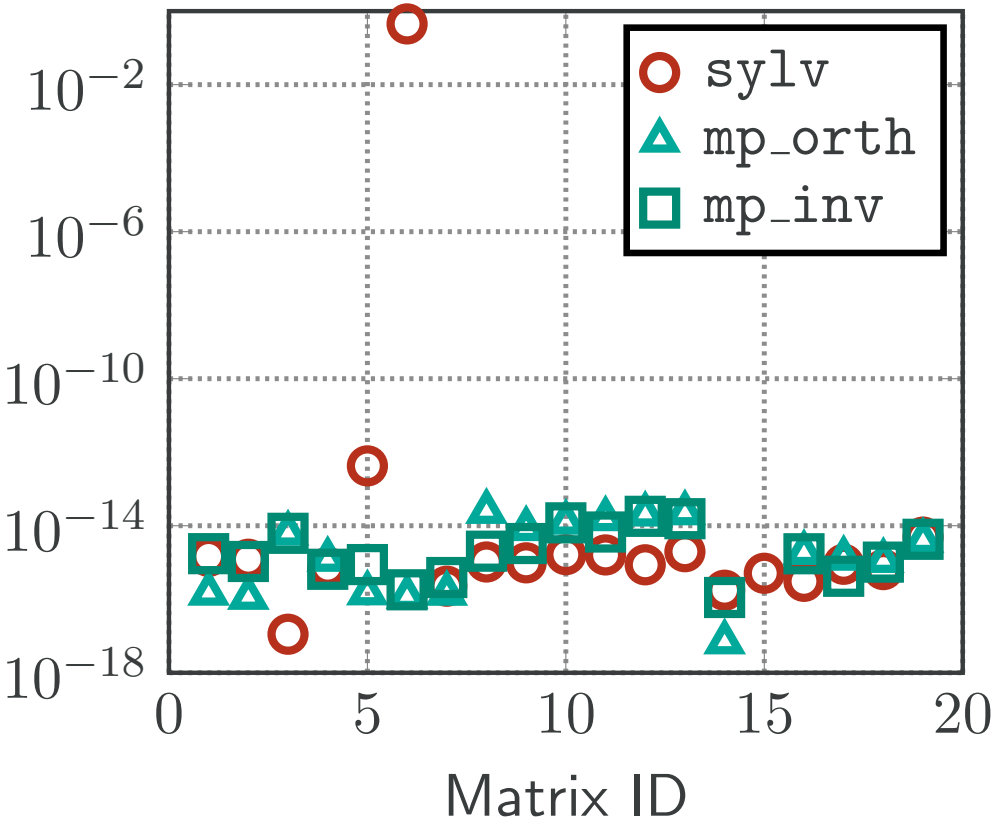
- 19 Sylvester equations from the literature, size 31–1,668

Precisions

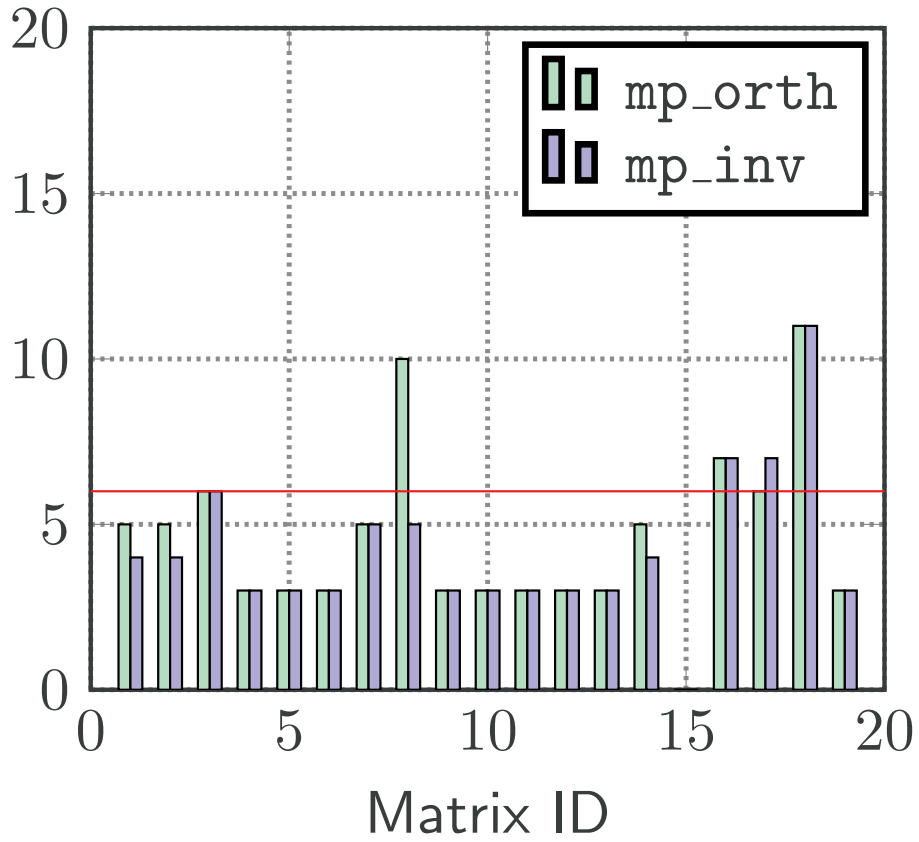
- TensorFloat-32, simulated by CPFloat (**Fasi & Mikaitis, '23**)
- Custom. 24-bit format: 16-bit signif., same range as TensorFloat-32
- binary64 (high precision)

Relative residual and number of iterations

Low-precision = TensorFloat-32



(c) Relative residual.

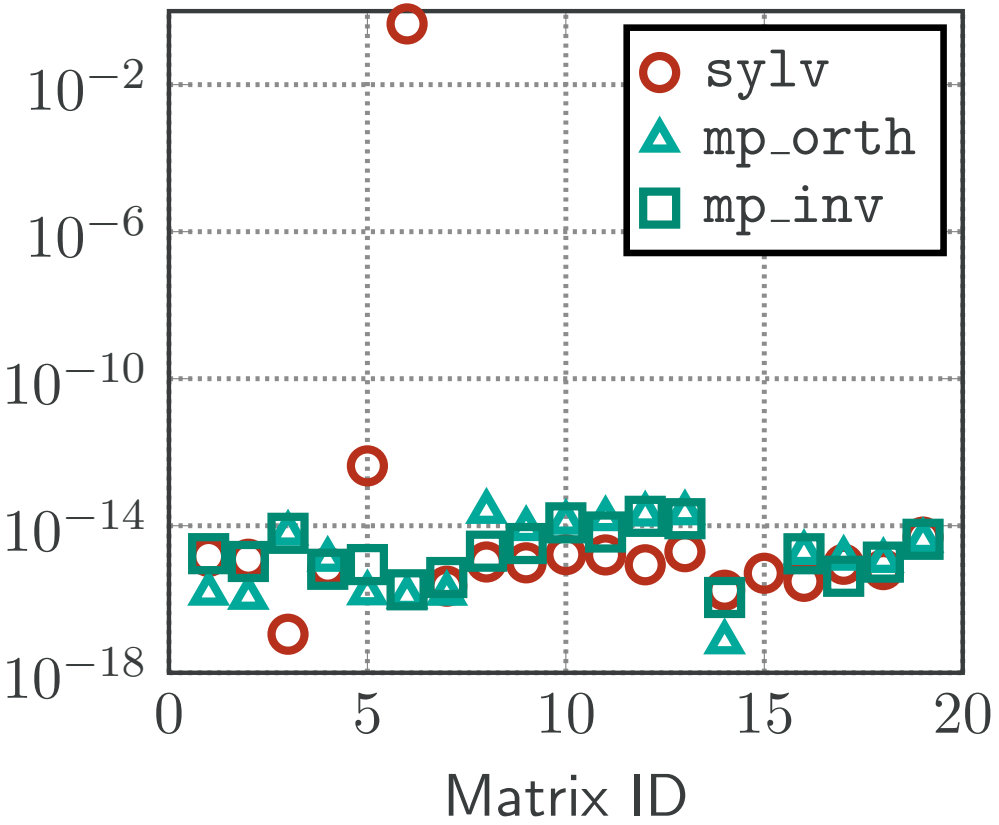


(d) Number of iterations.

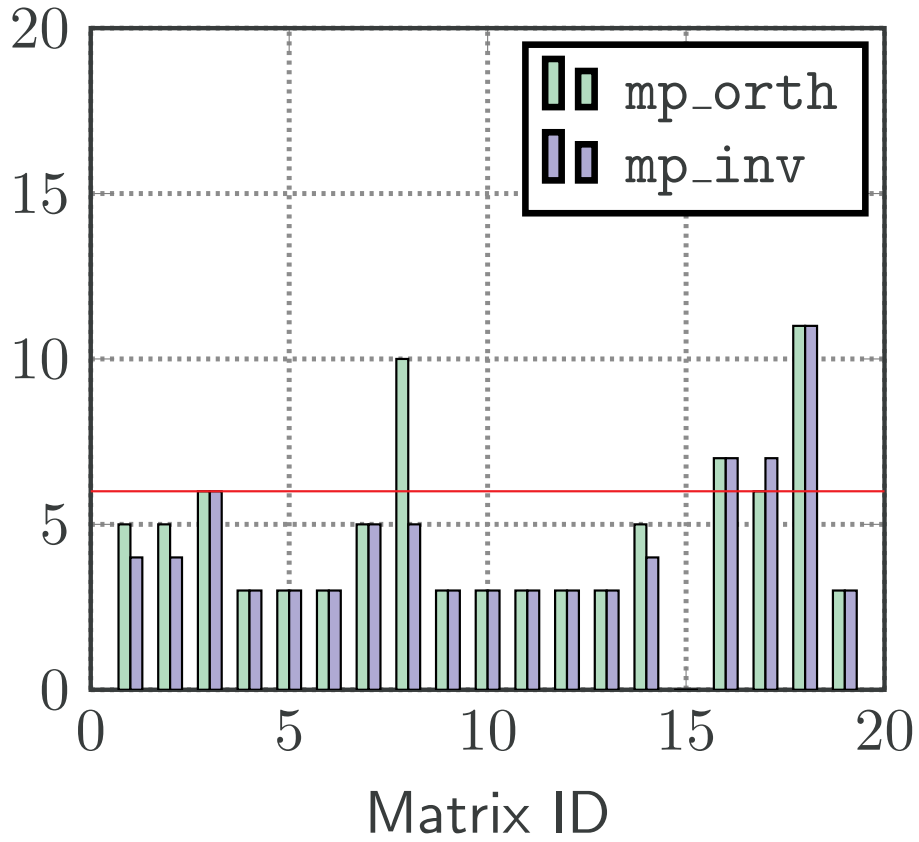
- Faster than Bartels–Stewart in most cases if $\rho \leq 0.2$ for tf32/fp64.
- tf32 $62.5\times$ faster on GB200 (180 PFLOPS/2880 TFLOPS) (**NVIDIA, '25**)

Relative residual and number of iterations

Low-precision = TensorFloat-32



(e) Relative residual.

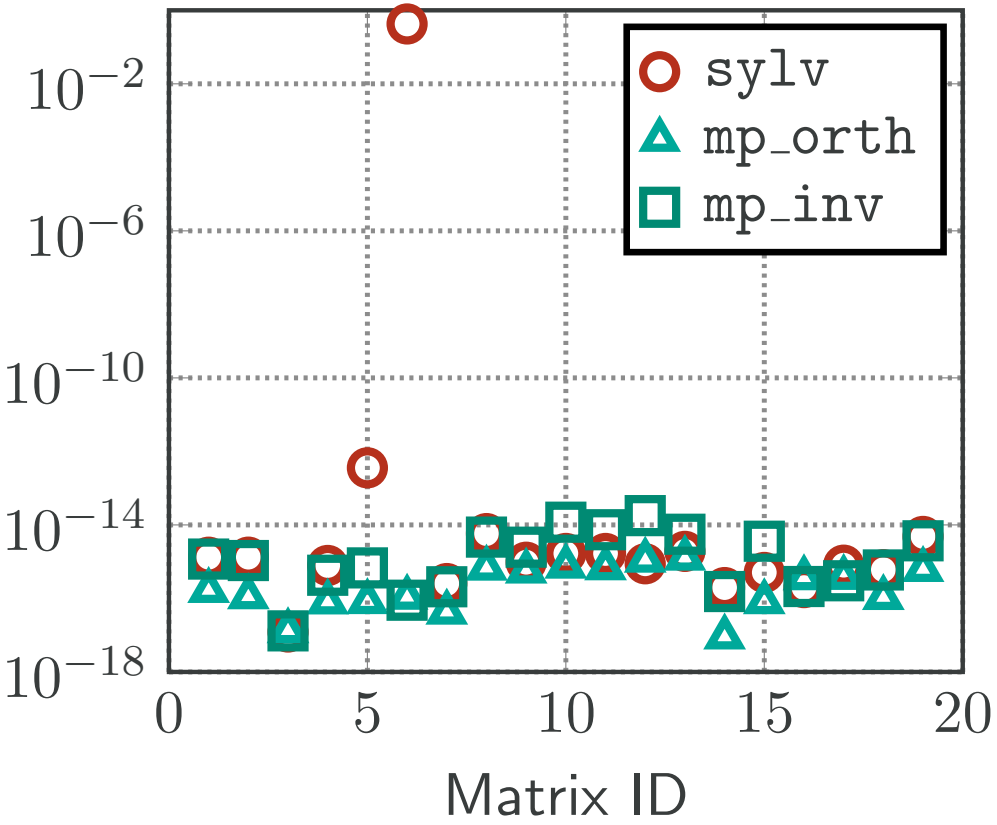


(f) Number of iterations.

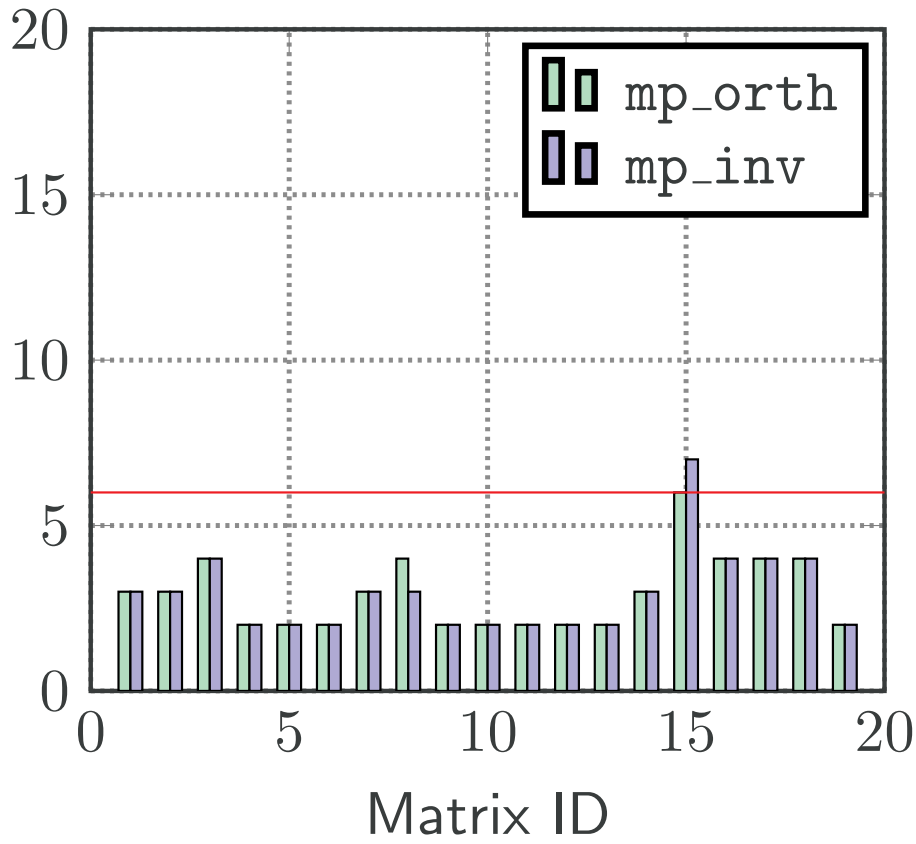
- Faster than Bartels–Stewart in most cases if $\rho \leq 0.2$ for tf32/fp64.
- tf32 $62.5\times$ faster on GB200 (180 PFLOPS/2880 TFLOPS) **(NVIDIA, '25)**

Relative residual and number of iterations

Low-precision = 24-bit custom. format



(g) Relative residual.

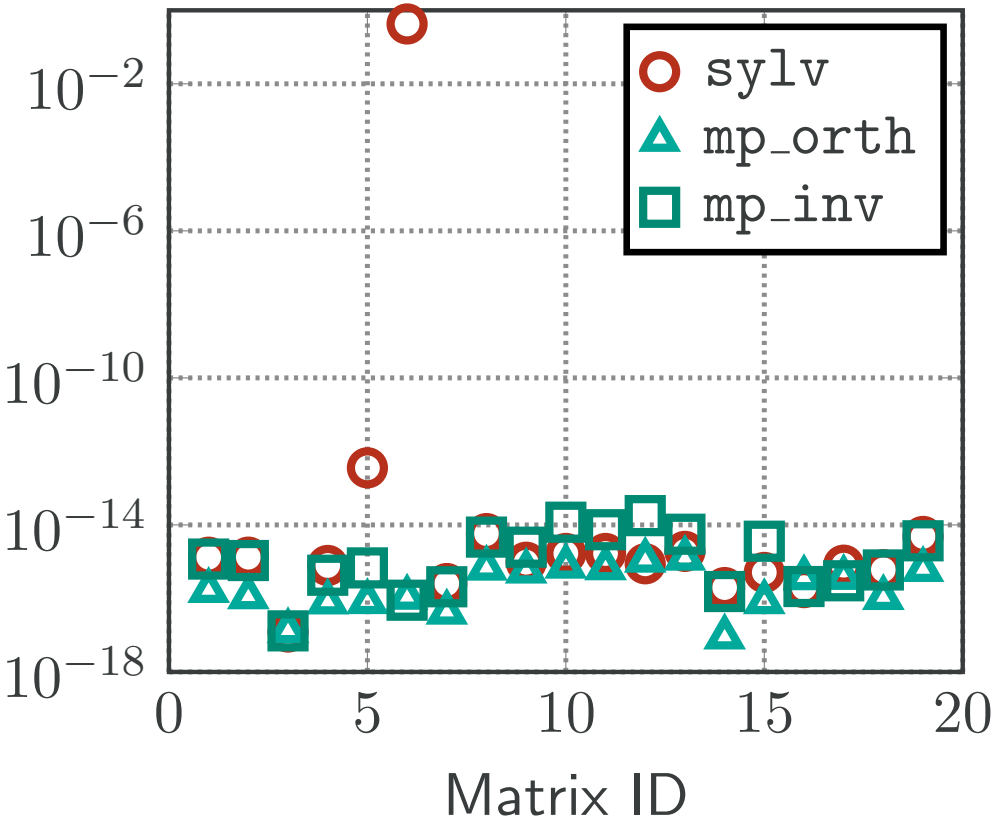


(h) Number of iterations.

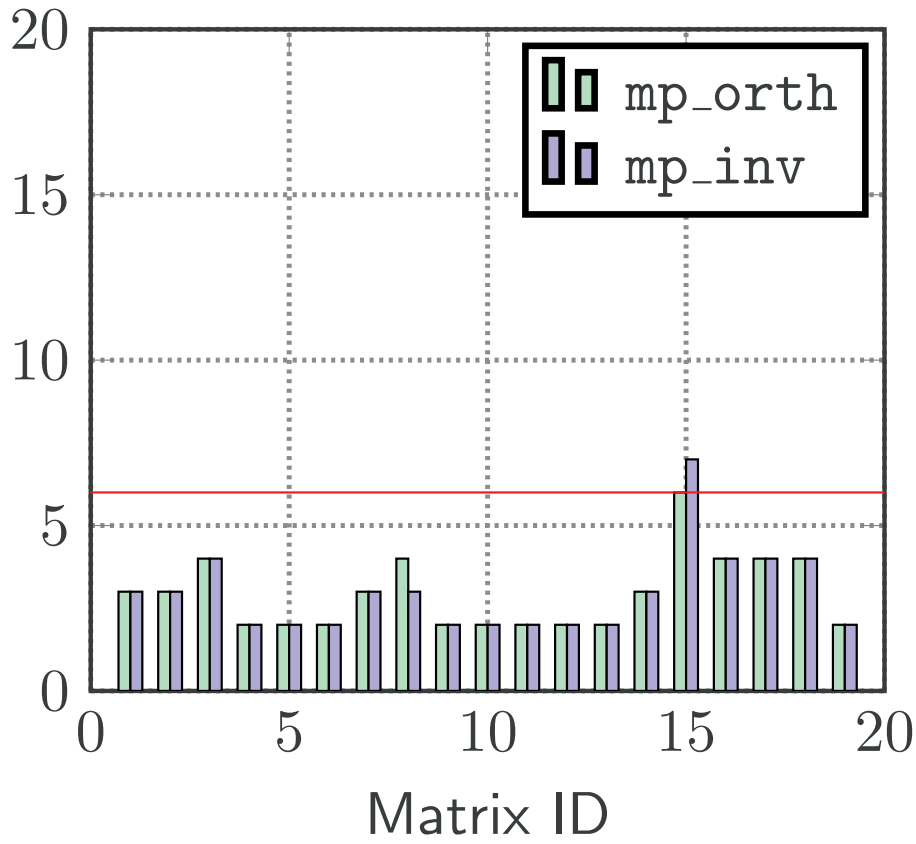
- New algorithms convergent in all cases
- Faster than Bartels–Stewart in most cases if $\rho \leq 0.4$ for 24-bit./fp64.

Relative residual and number of iterations

Low-precision = 24-bit custom. format



(i) Relative residual.



(j) Number of iterations.

- New algorithms convergent in all cases
- Faster than Bartels–Stewart in most cases if $\rho \leq 0.4$ for 24-bit./fp64.

Conclusions

Two mixed-precision algorithms for Sylvester equations

- Schur-based $\rightsquigarrow$ Probably more relevant for dense & moderate-sized
- Iterative refinement for high-precision sol'n to triangular equation
- Exploitation of easy inversion/re-ortho. of low-precision unitary matrix

Done, but not covered in the talk

- Analysis justifying the method & precision choice; converg. suff. cond
- Reduction to Lyapunov equation ($B = A^T$)

▶ A. Dmytryshyn, M. Fasi, N. J. Higham, and X. Liu. Mixed-precision algorithms for solving the Sylvester matrix equation. ArXiv:2503.03456 [math.NA], March 2025, https://arxiv.org/abs/2503.03456.

Next?

- High-performance fp16/bf16/tf32–fp64 implementations, when low-precision Schur (QR) decomposition available

Bibliography

R. H. Bartels and G. W. Stewart.
Algorithm 432: Solution of the Matrix Equation $AX + XB = C$.
Comm. ACM, 15(9):820–826, 1972.

M. Fasi and M. Mikaitis.
CPFloat: A C Library for Simulating Low-precision Arithmetic.
ACM Trans. Math. Software, 49(2):1–32, 2023.

NVIDIA Corporation.
NVIDIA Blackwell Architecture Technical Brief.
Tech. report, 2025.