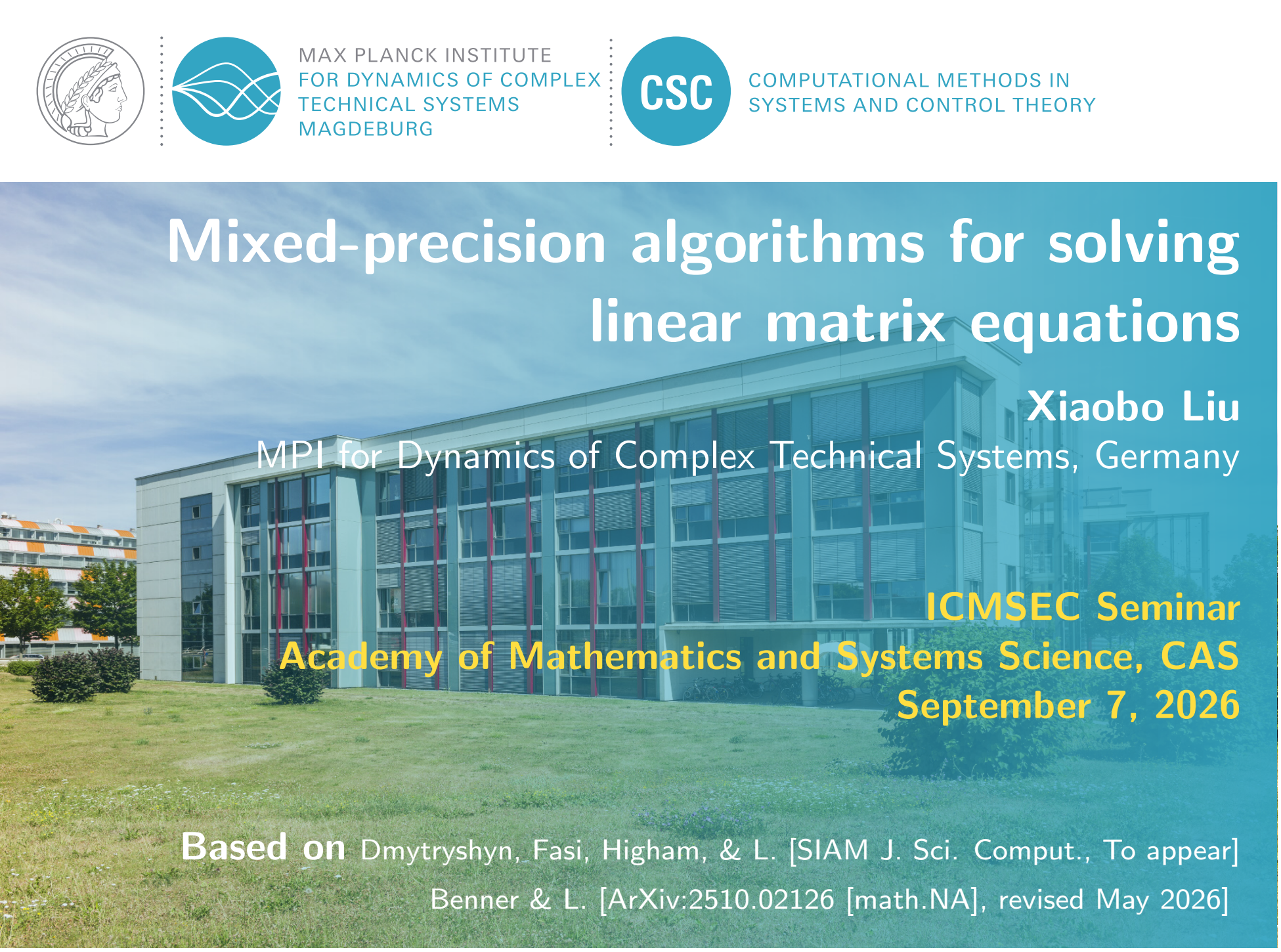


MAX PLANCK INSTITUTE
FOR DYNAMICS OF COMPLEX
TECHNICAL SYSTEMS
MAGDEBURG

CSC
COMPUTATIONAL METHODS IN
SYSTEMS AND CONTROL THEORY

# Mixed-precision algorithms for solving linear matrix equations

**Xiaobo Liu**
MPI for Dynamics of Complex Technical Systems, Germany

**ICMSEC Seminar**
**Academy of Mathematics and Systems Science, CAS**
**September 7, 2026**

**Based on** Dmytryshyn, Fasi, Higham, & L. [SIAM J. Sci. Comput., To appear]
Benner & L. [ArXiv:2510.02126 [math.NA], revised May 2026]

# Outline

1. **Brief overview on mixed precision**
2. Sylvester matrix equations
3. Low-rank Lyapunov matrix equations
4. Conclusions

# Floating-point arithmetics

Typical (normalized) floating-point format


$$x = (-1)^s \cdot (1.f_1 \cdots f_{t-1})_2 \cdot 2^{E-(2^{w-1}-1)}, \quad 2^{w-1}-1 \text{ the exponent bias}$$

Table: Parameters of four floating point formats.

| Type | Signif. bits ($t$) | Exp. bits ($w$) | Range | $u = 2^{-t}$ |
|---|---|---|---|---|
| bf16 | 8 | 8 | $10^{\pm 38}$ | $3.9 \times 10^{-3}$ |
| fp16 | 11 | 5 | $10^{\pm 5}$ | $4.9 \times 10^{-4}$ |
| fp32 | 24 | 8 | $10^{\pm 38}$ | $6.0 \times 10^{-8}$ |
| fp64 | 53 | 11 | $10^{\pm 308}$ | $1.1 \times 10^{-16}$ |

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

# Floating-point arithmetics

Typical (normalized) floating-point format


$$x = (-1)^s \cdot (1.f_1 \cdots f_{t-1})_2 \cdot 2^{E-(2^{w-1}-1)}, \quad 2^{w-1}-1 \text{ the exponent bias}$$

Table: Parameters of four floating point formats.

| Type | Signif. bits $(t)$ | Exp. bits $(w)$ | Range | $u = 2^{-t}$ |
|---|---|---|---|---|
| bf16 | 8 | 8 | $10^{\pm 38}$ | $3.9 \times 10^{-3}$ |
| fp16 | 11 | 5 | $10^{\pm 5}$ | $4.9 \times 10^{-4}$ |
| fp32 | 24 | 8 | $10^{\pm 38}$ | $6.0 \times 10^{-8}$ |
| fp64 | 53 | 11 | $10^{\pm 308}$ | $1.1 \times 10^{-16}$ |

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

# Floating-point arithmetics

Typical (normalized) floating-point format


$$x = (-1)^s \cdot (1.f_1 \cdots f_{t-1})_2 \cdot 2^{E-(2^{w-1}-1)}, \quad 2^{w-1}-1 \text{ the exponent bias}$$

Table: Parameters of four floating point formats.

| Type | Signif. bits $(t)$ | Exp. bits $(w)$ | Range | $u = 2^{-t}$ |
|---|---|---|---|---|
| bf16 | 8 | 8 | $10^{\pm 38}$ | $3.9 \times 10^{-3}$ |
| fp16 | 11 | 5 | $10^{\pm 5}$ | $4.9 \times 10^{-4}$ |
| fp32 | 24 | 8 | $10^{\pm 38}$ | $6.0 \times 10^{-8}$ |
| fp64 | 53 | 11 | $10^{\pm 308}$ | $1.1 \times 10^{-16}$ |

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

# Use of low and mixed precisions

When to use low precision?

- Only low accuracy needed
- To balance other sources of errors:
  model reduction, approximation (e.g., low rank, discretization)
- Self-correction mechanism available (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?

- Only low accuracy needed
- To balance other sources of errors:
  model reduction, approximation (e.g., low rank, discretization)
- Self-correction mechanism available (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?

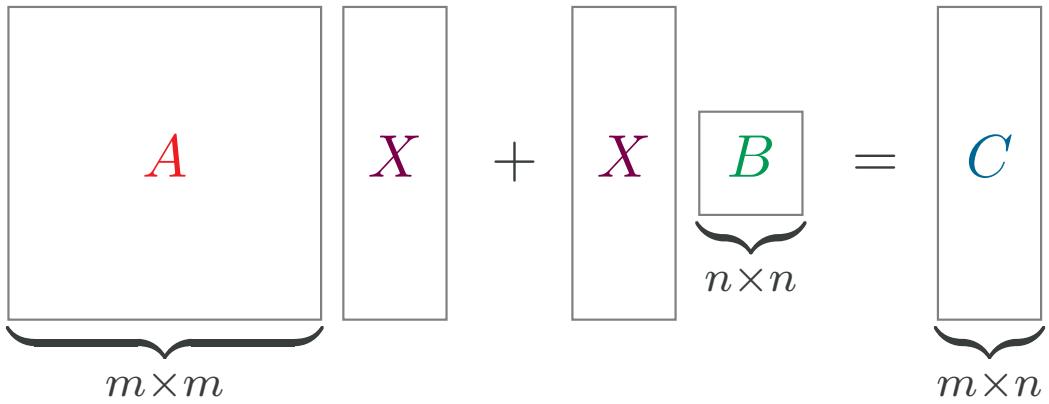
- Only low accuracy needed
- To balance other sources of errors:
  model reduction, approximation (e.g., low rank, discretization)
- Self-correction mechanism available (e.g., iterative refinement) ✓

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

# Outline

1. Brief overview on mixed precision
2. **Sylvester matrix equations**
3. Low-rank Lyapunov matrix equations
4. Conclusions

# The Sylvester matrix equations




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

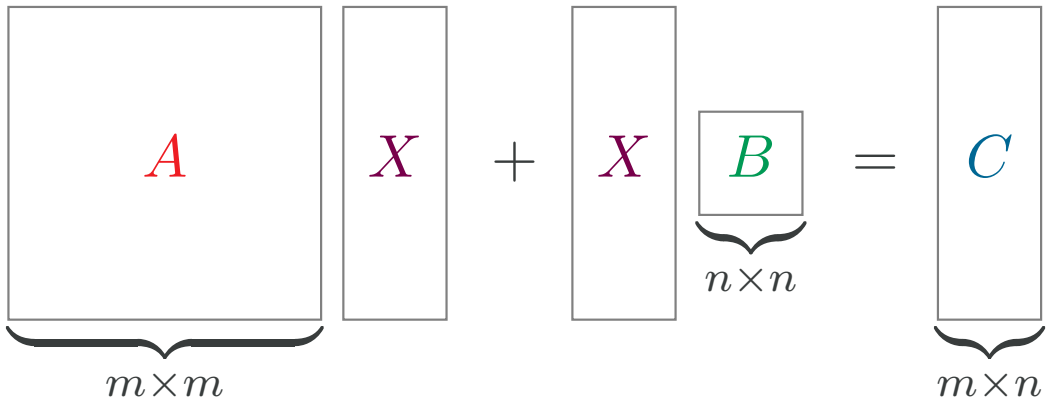
The setting:

- Dense coefficients of moderate size $\rightsquigarrow$ say, $\max(m, n) \lesssim 10^4$
- 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




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

The setting:

- Dense coefficients of moderate size $\rightsquigarrow$ say, $\max(m, n) \lesssim 10^4$
- 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 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 $X := U_A Y U_B^*$

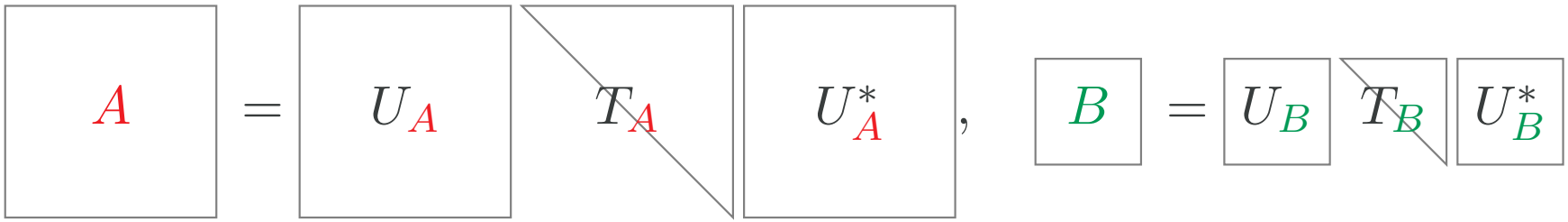
# The Bartels–Stewart algorithm

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

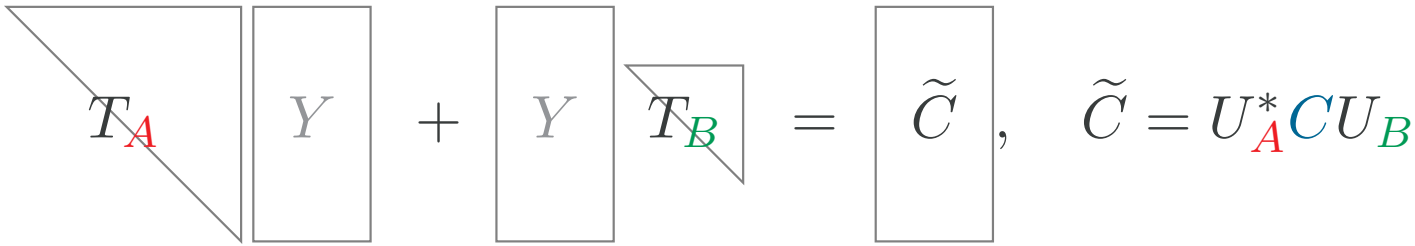
$$\Updownarrow$$

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

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:

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

2. Solve by substitution

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

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

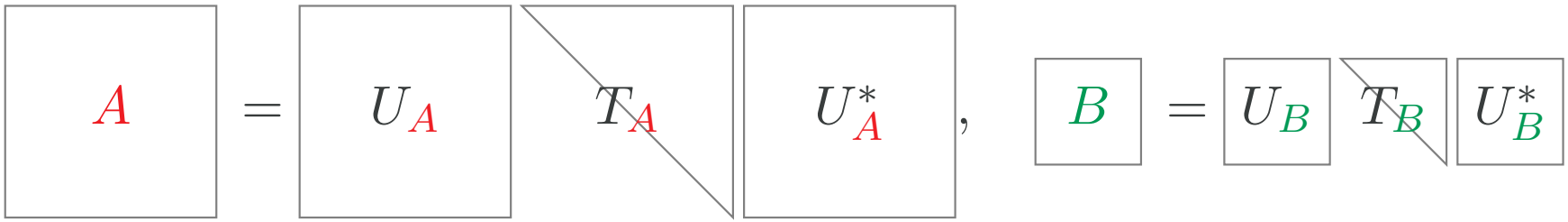
# The Bartels–Stewart algorithm

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

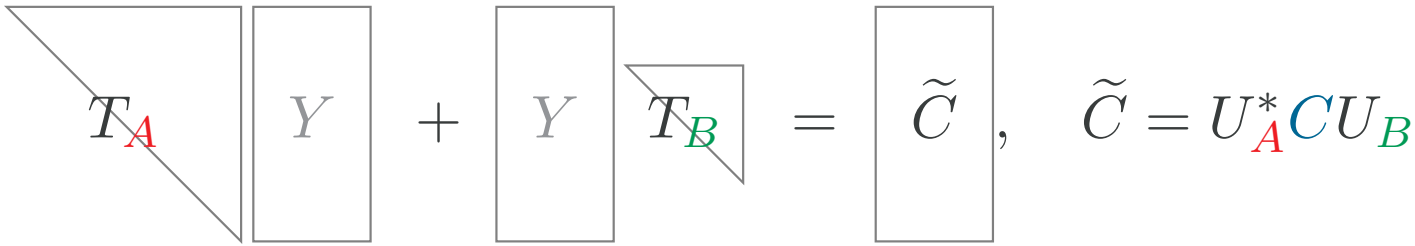
$$\Updownarrow$$

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

1. Comput. Schur decompositions:




2. Solve by substitution




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

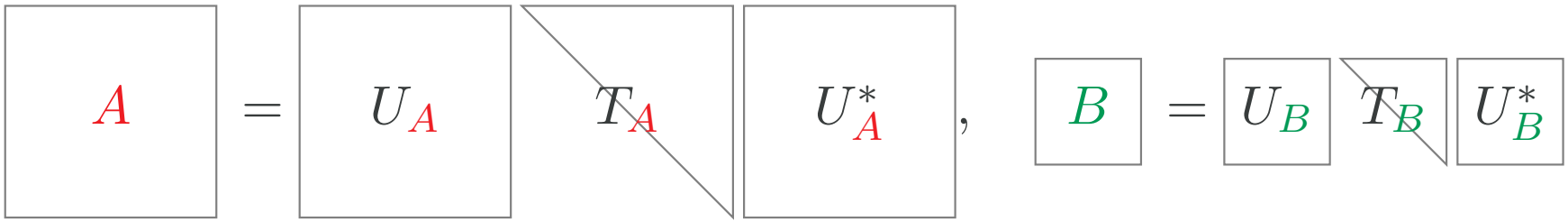
# The Bartels–Stewart algorithm

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

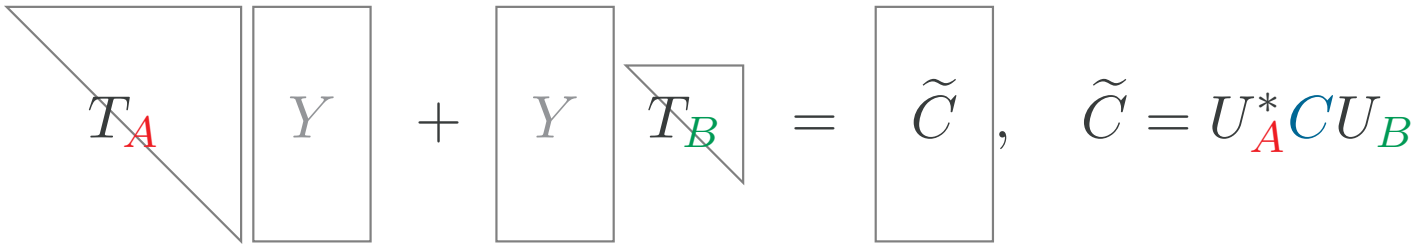
$$\Updownarrow$$

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

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 (IR) to solve a "near-by" equation to step 4.: coefficient matrices being low-precision triangular factors + small perturbation.
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 (IR) to solve a "near-by" equation to step 4.: coefficient matrices being low-precision triangular factors + small perturbation.
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 (IR) to solve a "near-by" equation to step 4.: coefficient matrices being low-precision triangular factors + small perturbation.
2. Re-orthonormalize or explicitly invert low-precision unitary factors

# IR for perturbed triangular equation

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

---

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

# IR for perturbed triangular equation

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

---

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

# IR for perturbed triangular equation

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

---

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 $\Delta_Y$ 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 $u_\ell$

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 $\Delta_Y$ 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 $u_\ell$

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 $\Delta_Y$ 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, e.g., GPUs

# 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, e.g., GPUs

# 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, e.g., GPUs

# 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
- $C_\ell$: number of low-precision flops

Each flop in $C_\ell$ is multiplied by $\rho$.

# How many iterations can we afford?

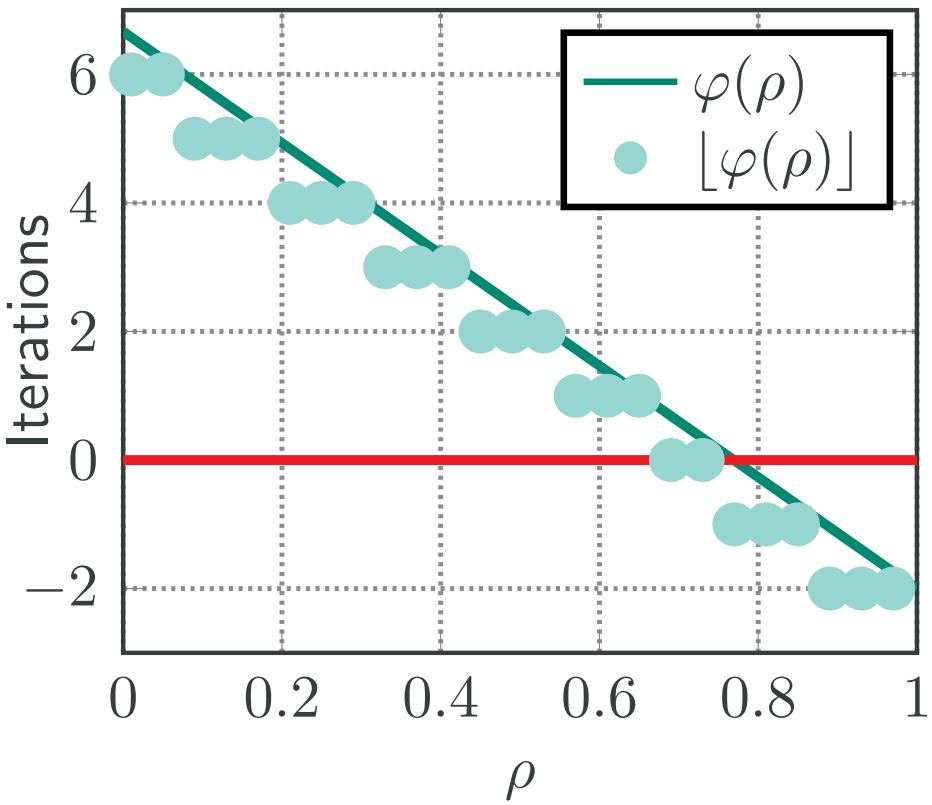
## Maximum number of iterations

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

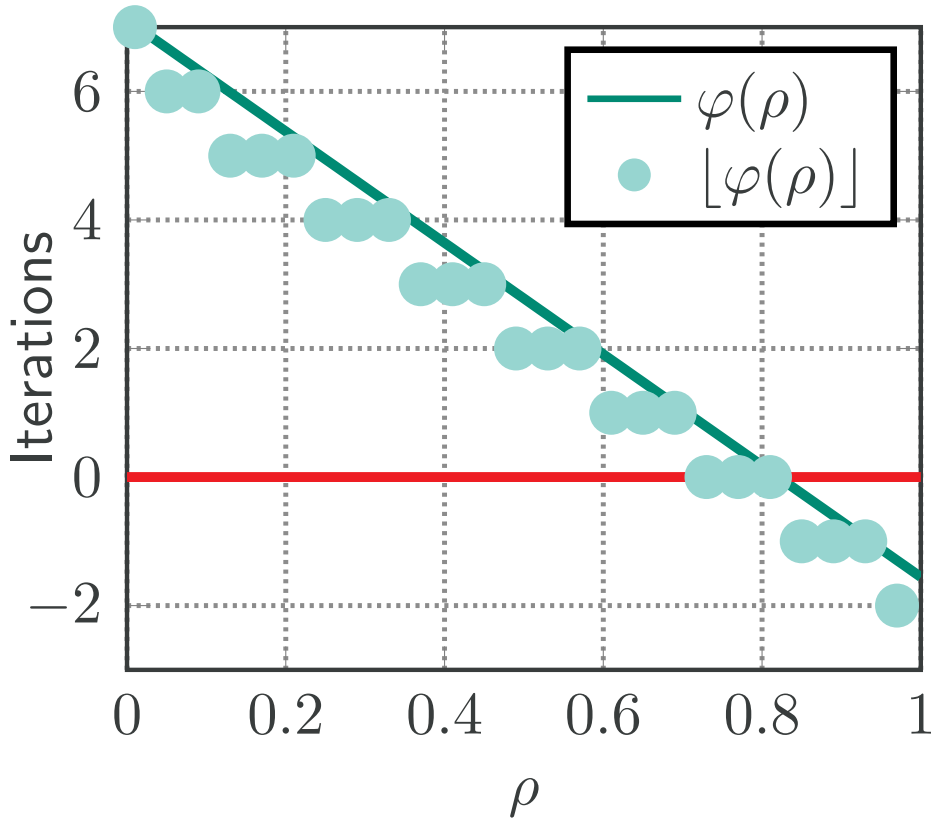
where

- $C_{BS}(m,n)$: 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 the refinement scheme

# At most $6$ iterations when $m = n$




(a) Re-orthonormalization-based.




(b) Inversion-based.

- $\rho =$ mean time of low-precision flop $/$ mean 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

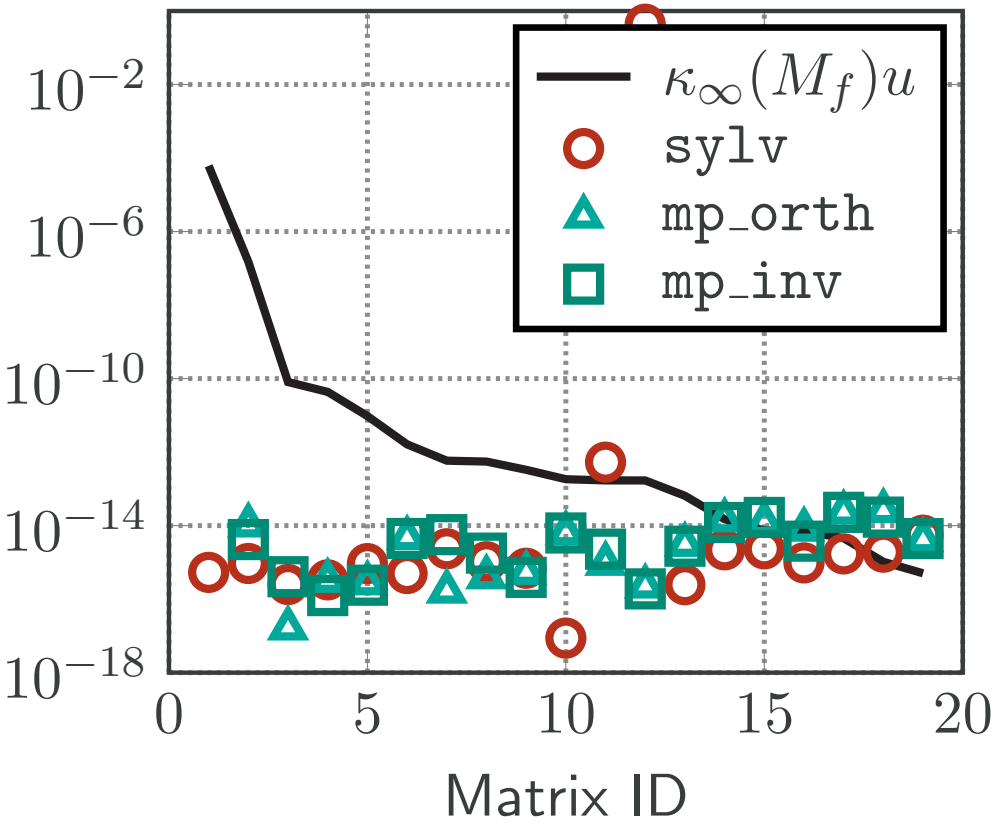
- 19 Sylvester equations from the literature, size 31–1,668

## Precisions

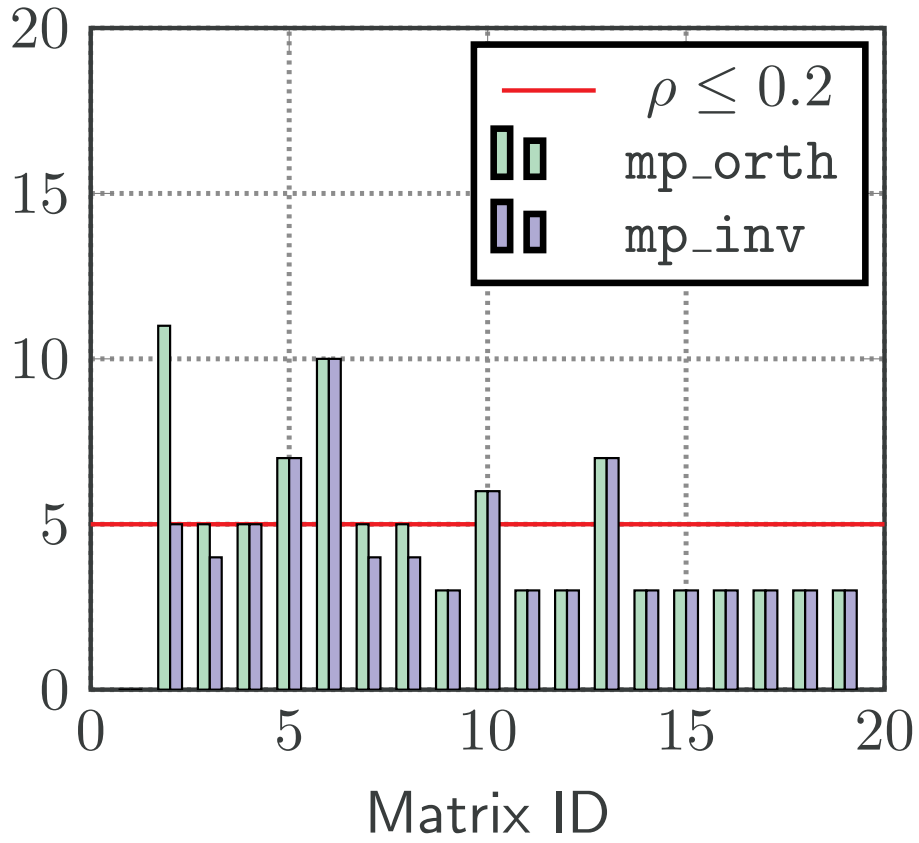
- TensorFloat-32, simulated by CPFloat (**Fasi & Mikaitis, '23**)
- Custom. 24-bit format: 16-bit signif., same range as TensorFloat-32
- binary64 (high precision)

# Relative residual and number of iterations

Low-precision = TensorFloat-32




(c) Relative residual.

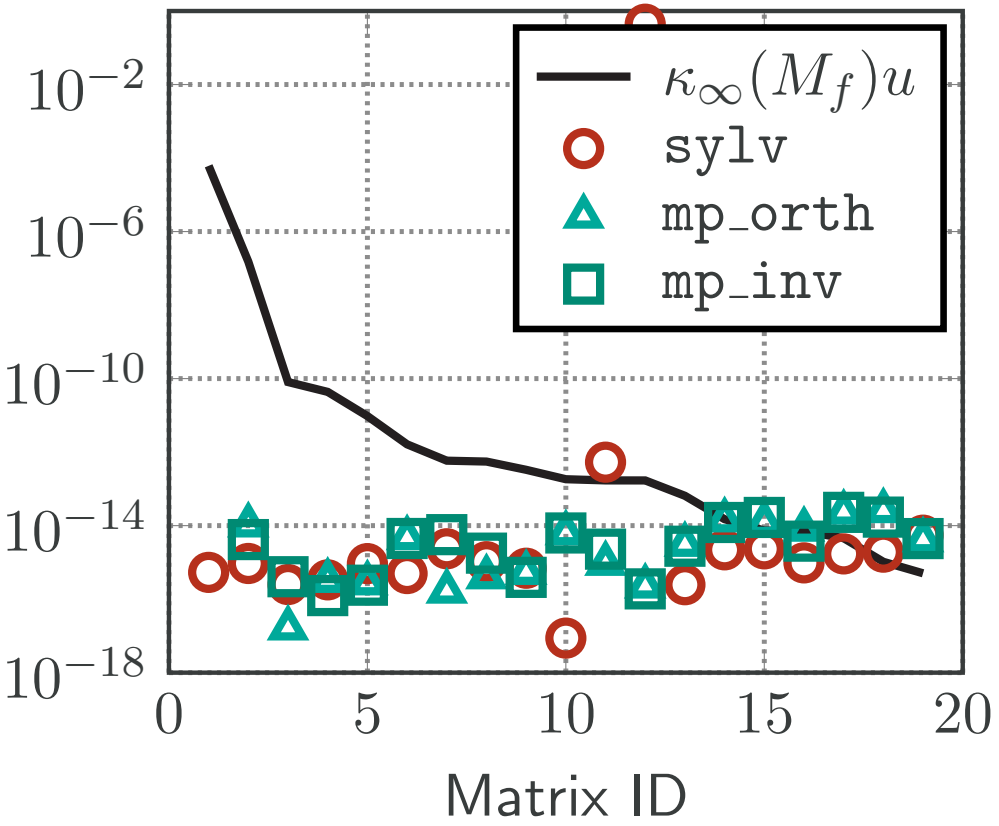



(d) Number of iterations.

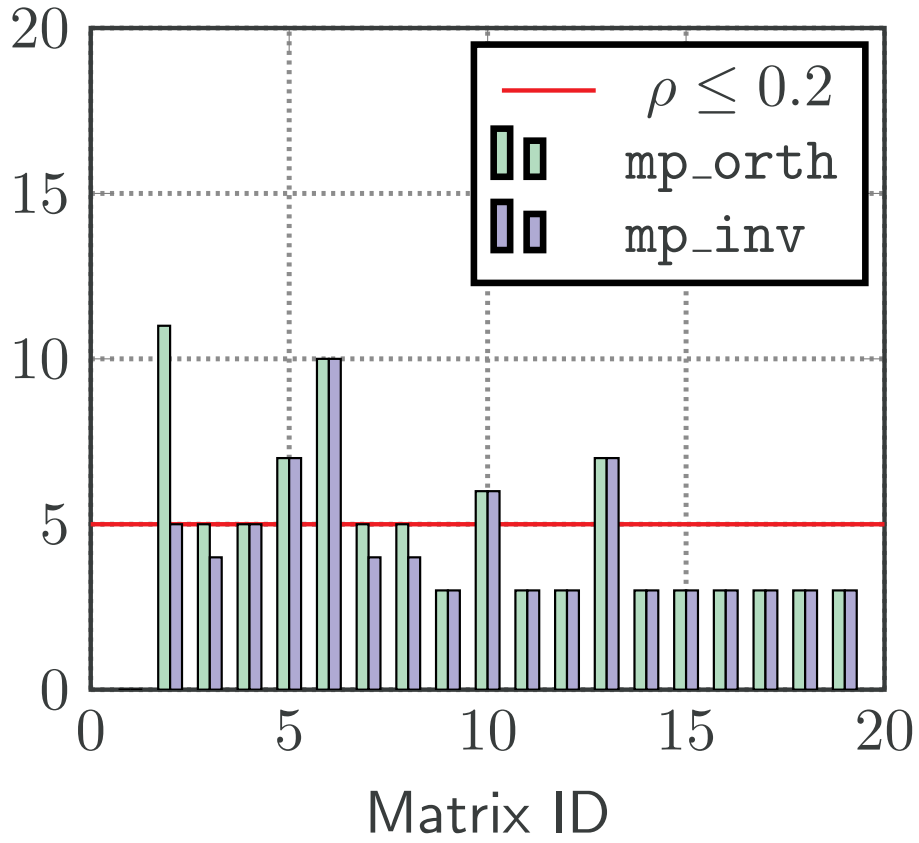
- Faster than Bartels–Stewart in most cases if $\rho \leq 0.2$ for tf32/fp64
- tf32 $31.3\times$ faster on GB200 TC (90 PFLOPS/2880 TFLOPS) (**NVIDIA, '25**)

# Relative residual and number of iterations

Low-precision = TensorFloat-32




(e) Relative residual.

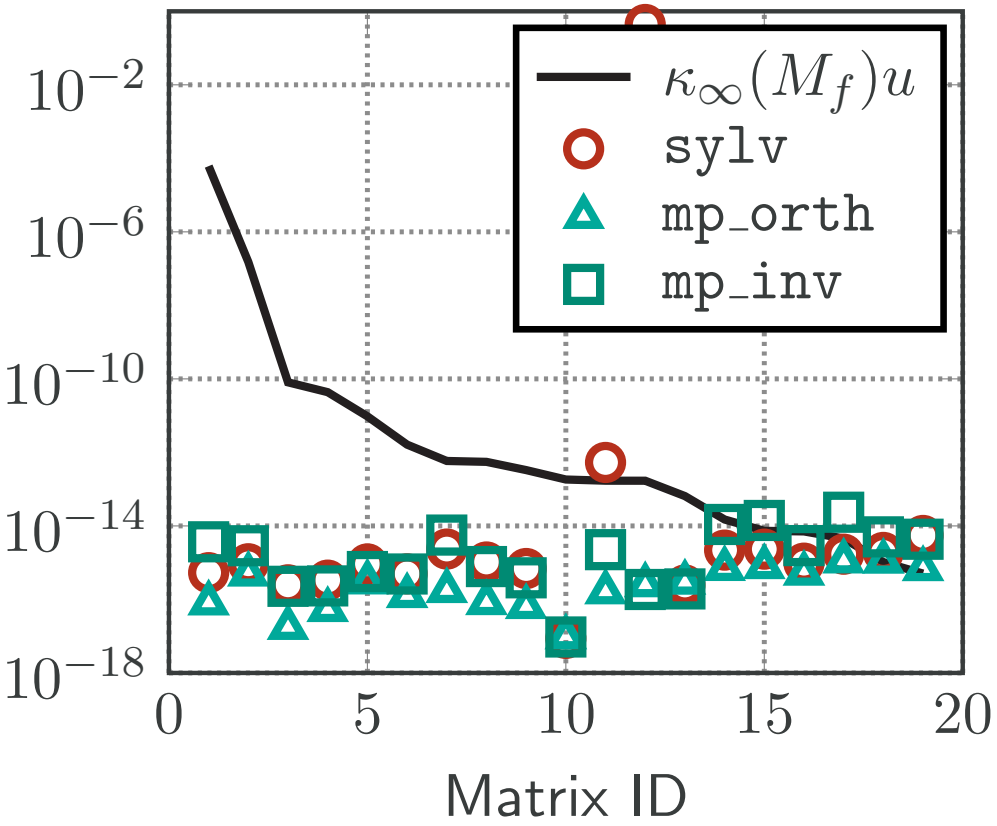



(f) Number of iterations.

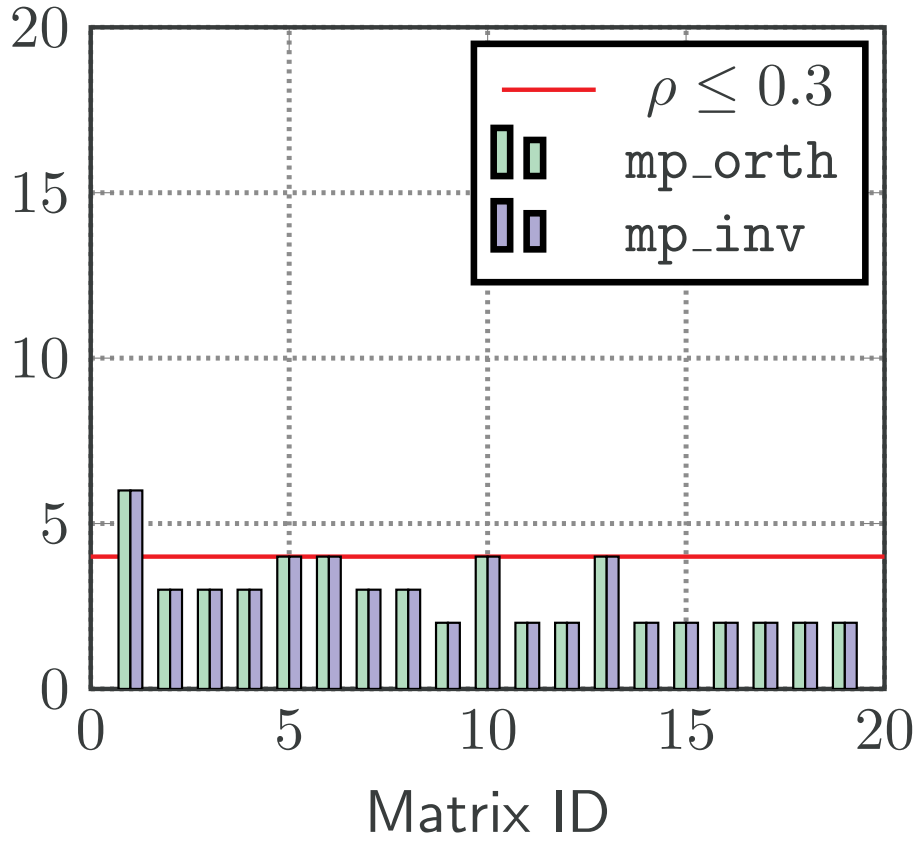
- Faster than Bartels–Stewart in most cases if $\rho \leq 0.2$ for tf32/fp64
- tf32 $31.3\times$ faster on GB200 TC (90 PFLOPS/2880 TFLOPS) **(NVIDIA, '25)**

# Relative residual and number of iterations

Low-precision = 24-bit custom. format




(g) Relative residual.

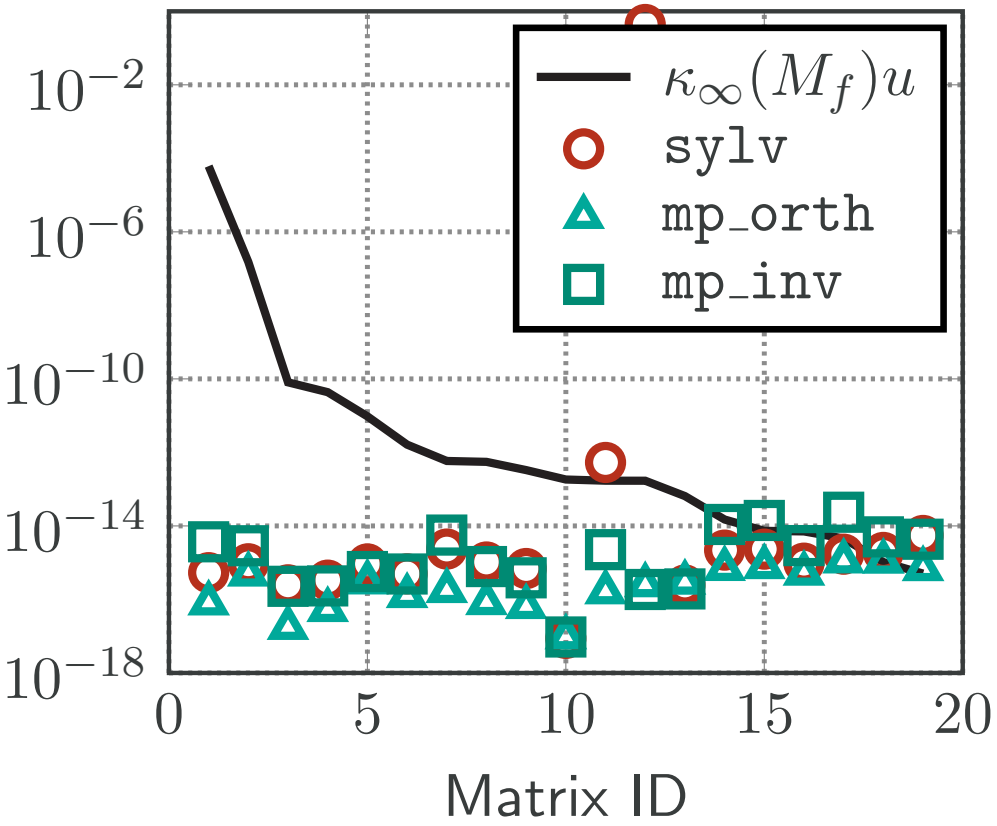



(h) Number of iterations.

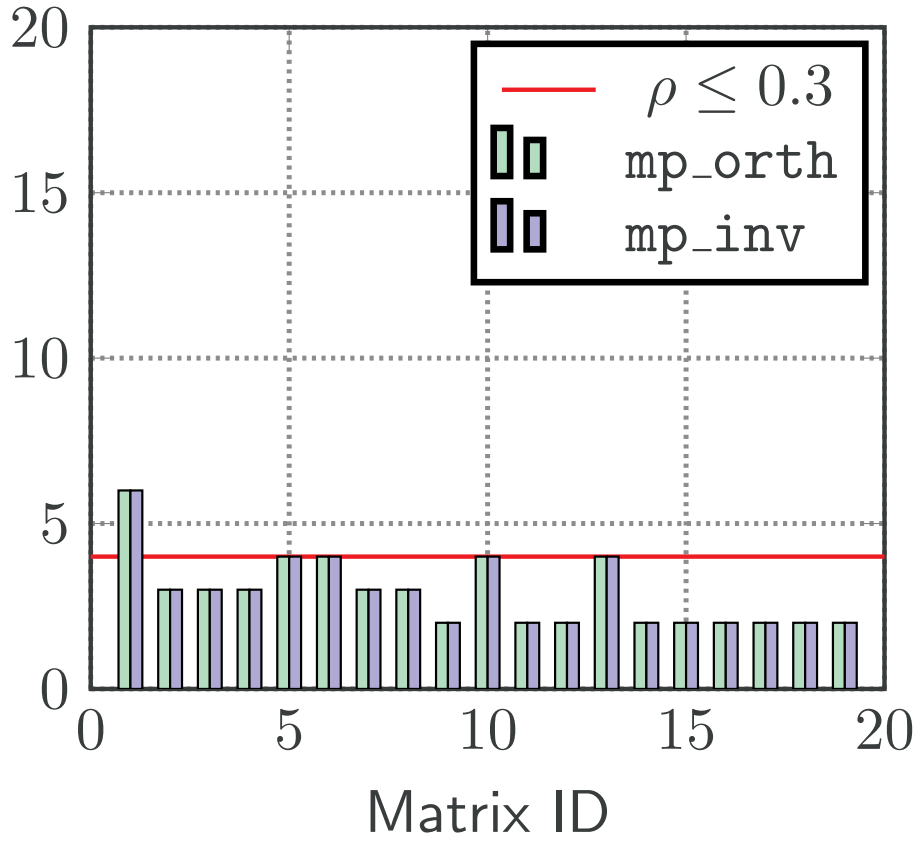
- New algorithms convergent in all cases
- Faster than Bartels–Stewart in most cases if $\rho \leq 0.3$ for 24-bit./fp64

# 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 $\rho \leq 0.3$ for 24-bit./fp64

# Outline

1. Brief overview on mixed precision
2. Sylvester matrix equations
3. **Low-rank Lyapunov matrix equations**
4. Conclusions

# The Lyapunov matrix equations

## Low-rank (algebraic) Lyapunov equations

$$\underbrace{A}_{n\times n} X + X A^T = -\underbrace{L L^T}_{W=LL^T} \quad \text{or} \quad -\underbrace{L S L^T}_{W=LSL^T}$$

$$L \in \mathbb{R}^{n\times m}, \qquad S = S^T \in \mathbb{R}^{m\times m}, \qquad m \ll n.$$

Applications in control theory, system balancing, and model reduction, ...

- $n \sim 10^4$ or beyond
- $\operatorname{Re}\lambda_i(A) < 0$, $W$ is PSD
- $X$ is PSD & has small numer. rank:

$$\|X - X_\varepsilon\| \le u\|X\|, \quad \operatorname{rank}(X_\varepsilon) \ll n$$

$\rightsquigarrow$ Look for $X = ZZ^T$, factor $Z \approx$ low-rank matrix (**Komaroff, '88**).

# The Lyapunov matrix equations

Low-rank (algebraic) Lyapunov equations

$$\underbrace{A}_{n\times n}X + X\underbrace{A^T}_{n\times n} = -\underbrace{L}_{W=LL^T}\underbrace{L^T}_{W=LL^T} \quad \text{or} \quad -\underbrace{L}_{W=LSL^T}\underbrace{S}_{W=LSL^T}\underbrace{L^T}_{W=LSL^T}$$

$$L \in \mathbb{R}^{n\times m}, \qquad S = S^T \in \mathbb{R}^{m\times m}, \qquad m \ll n.$$

Applications in control theory, system balancing, and model reduction, ...

- $n \sim 10^4$ or beyond
- $\operatorname{Re}\lambda_i(A) < 0$, $W$ is PSD
- $X$ is PSD & has small numer. rank:

$$\|X - X_\varepsilon\| \le u\|X\|, \quad \operatorname{rank}(X_\varepsilon) \ll n$$

$\rightsquigarrow$ Look for $X = ZZ^T$, factor $Z \approx$ low-rank matrix (**Komaroff, '88**).

# Iterative refinement for linear systems

## A mixed-precision framework for $Ax = b$

For $Ax = b$, use three prec. $u_r \leq u \leq u_s$: (**Carson & Higham, '18**)

- $u$, working precision
- $u_s$, solving correction equation
- $u_r$, residual computation

---

1 Solve $Ax = b$ in $u_s$ and store $x$ at $u$

2 **repeat**

3 Comput. $r = b - Ax$ at $u_r$ and round $r$ to $u_s$

4 Solve $Ad = r$ at $u_s$ and store $d$ at $u$

5 $x \leftarrow x + d$ at $u$.

6 **until** *converge*

---

- Backward error of $O(u)$, subject to $\kappa(A)$

# Adapt to low-rank Lyapunov equations

- Existing work: Solving $AX + XA^T + LL^T = 0$ in two-prec. iterative refinement (IR) (**Benner et al., '11**).
  - fp32-fp64 Implementation in hybrid CPU–GPU platforms
  - **Chol-type** solution factor $Z$ of $X = ZZ^T$
  - No rounding error analysis
- Our contributions:
  - Generalize to **LDL$^T$-type** sol'n factor $X = ZYZ^T$ $\rightsquigarrow$ allowing *indefinite* solution iterates
  - Mixed-precision error analysis for both types

# Adapt to low-rank Lyapunov equations

- Existing work: Solving $AX + XA^T + LL^T = 0$ in two-prec. iterative refinement (IR) (**Benner et al., '11**).
  - fp32-fp64 Implementation in hybrid CPU–GPU platforms
  - **Chol-type** solution factor $Z$ of $X = ZZ^T$
  - No rounding error analysis
- Our contributions:
  - Generalize to **LDL$^T$-type** sol'n factor $X = ZYZ^T$ $\rightsquigarrow$ allowing *indefinite* solution iterates
  - Mixed-precision error analysis for both types

# mpIR for low-rank Lyapunov equations

## LDL$^T$-type formulation

Comput. $X = ZYZ^T$ of $AX + XA^T + LSL^T = 0$.

---

1 Solve $AX + XA^T + LSL^T = 0$ in $u_s$ and store $Z$ and $Y$ at $u$

2 **repeat**

3 Comput. $\mathcal{R}(Z, Y) := AZYZ^T + ZYZ^TA^T + LSL^T = L_\Delta S_\Delta L_\Delta^T$ at $u_r$ and round $L_\Delta$ and $S_\Delta$ to $u_s$

4 Solve $AX_\Delta + X_\Delta A^T + L_\Delta S_\Delta L_\Delta^T = 0$ at $u_s$ and store $Z_\Delta$ and $Y_\Delta$ (of $X_\Delta = Z_\Delta Y_\Delta Z_\Delta^T$) at $u$

5 Find factors $Z$ and $Y$ of the *nearest* PSD matrix to $X \leftarrow X + X_\Delta$ at $u$. ▶ *RHS not PSD & $Y$, $Z$ obtained without forming $n \times n$ matrices*

6 **until** *converge*

---

- $O(n^2)$ flops in precisions higher than $u_s$: residual computation & solution update,

# mpIR for low-rank Lyapunov equations

## LDL$^T$-type formulation

Comput. $X = ZYZ^T$ of $AX + XA^T + LSL^T = 0$.

---

1 Solve $AX + XA^T + LSL^T = 0$ in $u_s$ and store $Z$ and $Y$ at $u$

2 **repeat**

3 Comput. $\mathcal{R}(Z, Y) := AZYZ^T + ZYZ^TA^T + LSL^T = L_\Delta S_\Delta L_\Delta^T$ at $u_r$ and round $L_\Delta$ and $S_\Delta$ to $u_s$

4 Solve $AX_\Delta + X_\Delta A^T + L_\Delta S_\Delta L_\Delta^T = 0$ at $u_s$ and store $Z_\Delta$ and $Y_\Delta$ (of $X_\Delta = Z_\Delta Y_\Delta Z_\Delta^T$) at $u$

5 Find factors $Z$ and $Y$ of the *nearest* PSD matrix to $X \leftarrow X + X_\Delta$ at $u$. ▶ *RHS not PSD & $Y$, $Z$ obtained without forming $n \times n$ matrices*

6 **until** *converge*

---

- $O(n^2)$ flops in precisions higher than $u_s$: residual computation & solution update,

# mpIR for low-rank Lyapunov equations

## LDL$^T$-type formulation

Comput. $X = ZYZ^T$ of $AX + XA^T + LSL^T = 0$.

---

1 Solve $AX + XA^T + LSL^T = 0$ in $u_s$ and store $Z$ and $Y$ at $u$

2 **repeat**

3 Comput. $\mathcal{R}(Z, Y) := AZYZ^T + ZYZ^TA^T + LSL^T = L_\Delta S_\Delta L_\Delta^T$ at $u_r$ and round $L_\Delta$ and $S_\Delta$ to $u_s$

4 Solve $AX_\Delta + X_\Delta A^T + L_\Delta S_\Delta L_\Delta^T = 0$ at $u_s$ and store $Z_\Delta$ and $Y_\Delta$ (of $X_\Delta = Z_\Delta Y_\Delta Z_\Delta^T$) at $u$

5 Find factors $Z$ and $Y$ of the *nearest* PSD matrix to $X \leftarrow X + X_\Delta$ at $u$. ▶ *RHS not PSD & $Y$, $Z$ obtained without forming $n \times n$ matrices*

6 **until** *converge*

---

- $O(n^2)$ flops in precisions higher than $u_s$: residual computation & solution update,

# mpIR for low-rank Lyapunov equations

## LDL$^T$-type formulation

Comput. $X = ZYZ^T$ of $AX + XA^T + LSL^T = 0$.

---

1 Solve $AX + XA^T + LSL^T = 0$ in $u_s$ and store $Z$ and $Y$ at $u$

2 **repeat**

3 Comput. $\mathcal{R}(Z, Y) := AZYZ^T + ZYZ^TA^T + LSL^T = L_\Delta S_\Delta L_\Delta^T$ at $u_r$ and round $L_\Delta$ and $S_\Delta$ to $u_s$

4 Solve $AX_\Delta + X_\Delta A^T + L_\Delta S_\Delta L_\Delta^T = 0$ at $u_s$ and store $Z_\Delta$ and $Y_\Delta$ (of $X_\Delta = Z_\Delta Y_\Delta Z_\Delta^T$) at $u$

5 Find factors $Z$ and $Y$ of the *nearest* PSD matrix to $X \leftarrow X + X_\Delta$ at $u$. ▶ *RHS not PSD & $Y$, $Z$ obtained without forming $n \times n$ matrices*

6 **until** *converge*

---

- $O(n^2)$ flops in precisions higher than $u_s$: residual computation & solution update,

# LDL$^T$-type residual decomposition

1. Rewrite $\mathcal{R}(Z,Y) = AZYZ^T + ZYZ^TA^T + LSL^T$ as

$$\mathcal{R}(Z,Y) = \begin{bmatrix} Z & AZ & L \end{bmatrix} \begin{bmatrix} 0 & Y & 0 \\ Y & 0 & 0 \\ 0 & 0 & S \end{bmatrix} \begin{bmatrix} Z^T \\ Z^TA^T \\ L^T \end{bmatrix}$$
$$=: FNF^T, \ F \text{ tall and skinny (as } Z \text{ and } L \text{ are)}.$$

2. Compute thin QR factorization $F = UT$, form $H = TNT^T$, and compute spectral decomposition $H = Q\Lambda Q^T$.

3. Perform eigenvalue truncation and drop $|\lambda_i| < \eta_r$,

$$\mathcal{R}(Z,Y) \approx UQ_t\Lambda_tQ_t^TU^T =: L_\Delta S_\Delta L_\Delta^T,$$

where

$$L_\Delta = UQ_t, \qquad S_\Delta = \Lambda_t.$$

# LDL$^T$-type residual decomposition

1. Rewrite $\mathcal{R}(Z,Y) = AZYZ^T + ZYZ^TA^T + LSL^T$ as

$$\mathcal{R}(Z,Y) = \begin{bmatrix} Z & AZ & L \end{bmatrix} \begin{bmatrix} 0 & Y & 0 \\ Y & 0 & 0 \\ 0 & 0 & S \end{bmatrix} \begin{bmatrix} Z^T \\ Z^TA^T \\ L^T \end{bmatrix}$$

$$=: FNF^T, \ F \text{ tall and skinny (as } Z \text{ and } L \text{ are)}.$$

2. Compute thin QR factorization $F = UT$, form $H = TNT^T$, and compute spectral decomposition $H = Q\Lambda Q^T$.

3. Perform eigenvalue truncation and drop $|\lambda_i| < \eta_r$,

$$\mathcal{R}(Z,Y) \approx UQ_t\Lambda_tQ_t^TU^T =: L_\Delta S_\Delta L_\Delta^T,$$

where

$$L_\Delta = UQ_t, \qquad S_\Delta = \Lambda_t.$$

# LDL$^T$-type residual decomposition

1. Rewrite $\mathcal{R}(Z,Y) = AZYZ^T + ZYZ^TA^T + LSL^T$ as

$$\mathcal{R}(Z,Y) = \begin{bmatrix} Z & AZ & L \end{bmatrix} \begin{bmatrix} 0 & Y & 0 \\ Y & 0 & 0 \\ 0 & 0 & S \end{bmatrix} \begin{bmatrix} Z^T \\ Z^TA^T \\ L^T \end{bmatrix}$$

$$=: FNF^T, \ F \text{ tall and skinny (as } Z \text{ and } L \text{ are)}.$$

2. Compute thin QR factorization $F = UT$, form $H = TNT^T$, and compute spectral decomposition $H = Q\Lambda Q^T$.

3. Perform eigenvalue truncation and drop $|\lambda_i| < \eta_r$,

$$\mathcal{R}(Z,Y) \approx UQ_t\Lambda_tQ_t^TU^T =: L_\Delta S_\Delta L_\Delta^T,$$

where

$$L_\Delta = UQ_t, \qquad S_\Delta = \Lambda_t.$$

# LDL$^T$-type solution factor update

Update $Z$ and $Y$ with $Z_\Delta$ and $Y_\Delta$.

1. Form

$$G := \begin{bmatrix} Z & Z_\Delta \end{bmatrix}, \; G \textit{ tall and skinny}, \textit{ and } \Upsilon = \begin{bmatrix} Y & \\ & Y_\Delta \end{bmatrix}.$$

2. Comput. *thin* QR factorization $G = V\Gamma$, form $K = \Gamma\Upsilon\Gamma^T$, and compute spectral decomposition $K = \Theta\Sigma\Theta^T$.

3. Perform eigenvalue truncation and drop $\sigma_i < \eta_s$,

$$Z_{\text{new}} = V\Theta_t, \quad Y_{\text{new}} = \Sigma_t,$$

such that

$$X_{\text{new}} \approx V\Theta_t\Sigma_t\Theta_t^T V^T = Z_{\text{new}} Y_{new} Z_{\text{new}}^T.$$

# LDL$^T$-type solution factor update

Update $Z$ and $Y$ with $Z_\Delta$ and $Y_\Delta$.

1. Form

$$G := \begin{bmatrix} Z & Z_\Delta \end{bmatrix}, \; G \textit{ tall and skinny}, \textit{ and } \Upsilon = \begin{bmatrix} Y & \\ & Y_\Delta \end{bmatrix}.$$

2. Comput. *thin* QR factorization $G = V\Gamma$, form $K = \Gamma\Upsilon\Gamma^T$, and compute spectral decomposition $K = \Theta\Sigma\Theta^T$.

3. Perform eigenvalue truncation and drop $\sigma_i < \eta_s$,

$$Z_{\text{new}} = V\Theta_t, \quad Y_{\text{new}} = \Sigma_t,$$

such that

$$X_{\text{new}} \approx V\Theta_t\Sigma_t\Theta_t^T V^T = Z_{\text{new}} Y_{new} Z_{\text{new}}^T.$$

# LDL$^T$-type solution factor update

Update $Z$ and $Y$ with $Z_\Delta$ and $Y_\Delta$.

1. Form

$$G := \begin{bmatrix} Z & Z_\Delta \end{bmatrix}, \; G \textit{ tall and skinny}, \textit{ and } \Upsilon = \begin{bmatrix} Y & \\ & Y_\Delta \end{bmatrix}.$$

2. Comput. *thin* QR factorization $G = V\Gamma$, form $K = \Gamma\Upsilon\Gamma^T$, and compute spectral decomposition $K = \Theta\Sigma\Theta^T$.

3. Perform eigenvalue truncation and drop $\sigma_i < \eta_s$,

$$Z_{\text{new}} = V\Theta_t, \quad Y_{\text{new}} = \Sigma_t,$$

such that

$$X_{\text{new}} \approx V\Theta_t\Sigma_t\Theta_t^T V^T = Z_{\text{new}} Y_{new} Z_{\text{new}}^T.$$

# Convergence bounds in two precisions

Table: Condition number bounds and limiting residuals, given a *stable* solver.

| $u_s$ | $u_r = u$ | Bound on $\kappa_F(\mathcal{L})$ | Limiting residual |
|---|---|---|---|
| bf16 | fp32 | $10^3$ | fp32 |
| fp16 | | $10^4$ | |
| fp32 | | $10^8$ | |
| bf16 | fp64 | $10^3$ | fp64 |
| fp16 | | $10^4$ | |
| fp32 | | $10^8$ | |
| fp64 | | $10^{16}$ | |

# Sign function Newton iteration

## As the solver in mpIR

- Sign function Newton iteration as an **LDL$^T$-type solver** (**Roberts, '80**):

$$AX + XA^T + W = 0, \quad W = LSL^T, \quad X = ZYZ^T.$$

  - Easy but robust implementation
  - Excellent parallelism
  - Rich in level-3 BLAS

- Other solvers are also applicable, provided **Chol/LDL$^T$-type** solution factors are returned

# Sign function Newton iteration

## Iterating on solution factors

For $\mu_{k-1} > 0$, iterate **(Larin & Aliev, '93; Benner & Quintana-Ortí, '99)**

$$A_k = \frac{1}{2}\left(\mu_{k-1} A_{k-1} + \mu_{k-1}^{-1} A_{k-1}^{-1}\right), \qquad A_0 = A,$$

$$Z_k = \begin{bmatrix} Z_{k-1} & A_{k-1}^{-1} Z_{k-1} \end{bmatrix}, \qquad Z_0 = L,$$

$$Y_k = \frac{1}{2}\begin{bmatrix} \mu_{k-1} Y_{k-1} & \\ & \mu_{k-1}^{-1} Y_{k-1} \end{bmatrix}, \qquad Y_0 = S.$$

At convergence,

$$\frac{1}{2} Z_k Y_k Z_k^T \longrightarrow X, \qquad Z = Z_k, \quad Y = \frac{1}{2} Y_k,$$

and

$$A_k \longrightarrow -I.$$

# Sign function Newton iteration

## Iterating on solution factors

For $\mu_{k-1} > 0$, iterate **(Larin & Aliev, '93; Benner & Quintana-Ortí, '99)**

$$\begin{aligned} A_k &= \frac{1}{2}\left(\mu_{k-1} A_{k-1} + \mu_{k-1}^{-1} A_{k-1}^{-1}\right), & A_0 &= A, \\ Z_k &= \begin{bmatrix} Z_{k-1} & A_{k-1}^{-1} Z_{k-1} \end{bmatrix}, & Z_0 &= L, \\ Y_k &= \frac{1}{2}\begin{bmatrix} \mu_{k-1} Y_{k-1} & \\ & \mu_{k-1}^{-1} Y_{k-1} \end{bmatrix}, & Y_0 &= S. \end{aligned}$$

At convergence,

$$\frac{1}{2} Z_k Y_k Z_k^T \longrightarrow X, \qquad Z = Z_k, \quad Y = \frac{1}{2} Y_k,$$

and

$$A_k \longrightarrow -I.$$

# Sign function Newton iteration
## Rank truncation

- In each iteration, numbers of columns of $Z_k$ and $Y_k$ double, leading to exponential storage growth:

$$
\begin{aligned}
A_k &= \frac{1}{2}\left(\mu_{k-1}A_{k-1} + \mu_{k-1}^{-1}A_{k-1}^{-1}\right), && A_0 = A,\\
Z_k &= \begin{bmatrix} Z_{k-1} & A_{k-1}^{-1}Z_{k-1} \end{bmatrix}, && Z_0 = L,\\
Y_k &= \begin{bmatrix} \frac{\mu_{k-1}}{2}Y_{k-1} & \\ & \frac{1}{2\mu_{k-1}}Y_{k-1} \end{bmatrix}, && Y_0 = S.
\end{aligned}
$$

$\rightsquigarrow$ Resolution: *rank truncation* on $Z_k$ and $Y_k$ (**Benner & Quintana-Ortí, '99**):

**Subfunction** $[Z,\ Y] = \textsc{RankTruncLdlt}(Z,\ Y)$

1. Compute thin QR factorization $Z = QR$.
2. Compute spectral decomposition $RYR^T = \widetilde{V}\widetilde{\Lambda}\widetilde{V}^T$.
3. $\Lambda = \operatorname{diag}(\widetilde{\lambda}_i)$, $i \in I_\lambda := \{i \mid \widetilde{\lambda}_i > u\|\widetilde{\Lambda}\|_1\}$ and $V = \widetilde{V}(:, I_\lambda)$
4. $Z = QV$, $Y = \Lambda$

# Sign function Newton iteration
## Rank truncation

- In each iteration, numbers of columns of $Z_k$ and $Y_k$ double, leading to exponential storage growth:

$$
\begin{aligned}
A_k &= \frac{1}{2}\left(\mu_{k-1}A_{k-1} + \mu_{k-1}^{-1}A_{k-1}^{-1}\right), && A_0 = A,\\
Z_k &= \begin{bmatrix} Z_{k-1} & A_{k-1}^{-1}Z_{k-1} \end{bmatrix}, && Z_0 = L,\\
Y_k &= \begin{bmatrix} \frac{\mu_{k-1}}{2}Y_{k-1} & \\ & \frac{1}{2\mu_{k-1}}Y_{k-1} \end{bmatrix}, && Y_0 = S.
\end{aligned}
$$

$\rightsquigarrow$ Resolution: *rank truncation* on $Z_k$ and $Y_k$ (**Benner & Quintana-Ortí, '99**):

---

**Subfunction** $[Z,\ Y] = \textsc{RankTruncLdlt}(Z,\ Y)$

1 Compute thin QR factorization $Z = QR$.

2 Compute spectral decomposition $RYR^T = \widetilde{V}\widetilde{\Lambda}\widetilde{V}^T$.

3 $\Lambda = \operatorname{diag}(\widetilde{\lambda}_i)$, $i \in I_\lambda := \{i \mid \widetilde{\lambda}_i > u\|\widetilde{\Lambda}\|_1\}$ and $V = \widetilde{V}(:, I_\lambda)$

4 $Z = QV$, $Y = \Lambda$

---

# Sign function Newton iteration

## The full algorithm

**Algorithm:** LDL$^T$-type sign function Newton iteration for the Lyapunov equation $AX + XA^T + W = 0$, $W = LSL^T$.

---

$Z_0 = L$, $Y_0 = S$, $A_0 = A$
**for** $k \leftarrow 1$ **to** $k_{\max}$ **do**
  $A_k = \frac{1}{2}(\mu_{k-1} A_{k-1} + \mu_{k-1}^{-1} A_{k-1}^{-1})$ [*] with $\mu_{k-1}$ a form of scaling.
  $Z_k = \begin{bmatrix} Z_{k-1} & A_{k-1}^{-1} Z_{k-1} \end{bmatrix}$
  $Y_k = \frac{1}{2}\,\mathrm{blkdiag}(\mu_{k-1} Y_{k-1},\ \mu_{k-1}^{-1} Y_{k-1})$
  **if** $\mathrm{size}(Z_k, 2) > \rho n$ **then** $[Z_k,\ Y_k] = \textsc{RankTruncLdlt}(Z_k,\ Y_k)$
  **if** $\|A_k + I_n\|_1 \leq \tau$ **then**
    Terminate after two more iterations.
$Y = Y_k/2$

---

- Parameters: convergence tol: $\tau = 10\sqrt{nu}$, rank truncation tol: $\rho \ll 1$
- [*]Dominant cost each iteration: $2n^3$

# Iterative refinement for Lyapunov equations
## Computational cost analysis

In each refinement (outer) step:

- Three main steps
- $O(n^2)$ flops for residual decomposition and solution factor update
- Assume a flop at $u_r$ or $u$ is no more expensive than $n$ flops at $u_s$
- Dominant cost (solver step): $2kn^3$ for $k$ Newton (inner) iterations

- Dominant cost: $2(k_0 + k_1 + \cdots + k_i)n^3$ flops in $u_s$. $k_\ell$ the number of Newton (inner) iterations at $\ell$th refinement step.

- A cost balance: lower precision $u_s \rightsquigarrow$ slower convergence of Newton (inner) iteration, $k_i$ grows.

# Iterative refinement for Lyapunov equations
## Computational cost analysis

In each refinement (outer) step:

- Three main steps
- $O(n^2)$ flops for residual decomposition and solution factor update
- Assume a flop at $u_r$ or $u$ is no more expensive than $n$ flops at $u_s$
- Dominant cost (solver step): $2kn^3$ for $k$ Newton (inner) iterations

• Dominant cost: $2(k_0 + k_1 + \cdots + k_i)n^3$ flops in $u_s$. $k_\ell$ the number of Newton (inner) iterations at $\ell$th refinement step.

• A cost balance: lower precision $u_s \rightsquigarrow$ slower convergence of Newton (inner) iteration, $k_i$ grows.

# Iterative refinement for Lyapunov equations
## Computational cost analysis

In each refinement (outer) step:

- Three main steps
- $O(n^2)$ flops for residual decomposition and solution factor update
- Assume a flop at $u_r$ or $u$ is no more expensive than $n$ flops at $u_s$
- Dominant cost (solver step): $2kn^3$ for $k$ Newton (inner) iterations

• Dominant cost: $2(k_0 + k_1 + \cdots + k_i)n^3$ flops in $u_s$. $k_\ell$ the number of Newton (inner) iterations at $\ell$th refinement step.

• A cost balance: lower precision $u_s \rightsquigarrow$ slower convergence of Newton (inner) iteration, $k_i$ grows.

# Iterative refinement for Lyapunov equations

## Alternative cost model

- Fact: in each refinement (outer) step, same sequence generated:

$$A_1^{-1}, A_2^{-1}, \ldots, \quad A_0 = A.$$

$\rightsquigarrow$ **Idea**: reuse $\widehat{A}_1^{-1}, \widehat{A}_2^{-1}, \ldots$ across different refinement iterations

- Dominant cost: ~~$2\sum_{\ell=0}^{i} k_\ell n^3$~~ $2\max_{0\le\ell\le i} k_\ell n^3$ flops in precision $u_s$. $k_\ell$ the number of Newton (inner) iterations at $\ell$th refinement step.

- Reduced computational cost & additional storage requirement: particularly interesting when $\widehat{A}_1^{-1}, \widehat{A}_2^{-1}, \ldots$ fits into *fast memory*

**Quantity of interest** in computational cost comparison:

$$k^\sigma =: \sum_{\ell=0}^{i} k_\ell \quad \text{or} \quad k^{\max} =: \max_{0\le\ell\le i} k_\ell.$$

Compare computational costs for mpIR with different solver precisions:

$$k^\sigma_{\text{low}}/k^\sigma_{\text{high}} \quad \text{or} \quad k^{\max}_{\text{low}}/k^{\max}_{\text{high}}$$

# Iterative refinement for Lyapunov equations

## Alternative cost model

- Fact: in each refinement (outer) step, same sequence generated:

$$A_1^{-1}, A_2^{-1}, \ldots, \quad A_0 = A.$$

$\rightsquigarrow$ **Idea**: reuse $\widehat{A}_1^{-1}, \widehat{A}_2^{-1}, \ldots$ across different refinement iterations

- Dominant cost: ~~$2\sum_{\ell=0}^{i} k_\ell n^3$~~ $2 \max_{0\le\ell\le i} k_\ell n^3$ flops in precision $u_s$. $k_\ell$ the number of Newton (inner) iterations at $\ell$th refinement step.

- Reduced computational cost & additional storage requirement: particularly interesting when $\widehat{A}_1^{-1}, \widehat{A}_2^{-1}, \ldots$ fits into *fast memory*

**Quantity of interest** in computational cost comparison:

$$k^\sigma =: \sum_{\ell=0}^{i} k_\ell \quad \text{or} \quad k^{\max} =: \max_{0\le\ell\le i} k_\ell.$$

Compare computational costs for mpIR with different solver precisions:

$$k^\sigma_{\text{low}}/k^\sigma_{\text{high}} \quad \text{or} \quad k^{\max}_{\text{low}}/k^{\max}_{\text{high}}$$

# Iterative refinement for Lyapunov equations

## Alternative cost model

- Fact: in each refinement (outer) step, same sequence generated:

$$A_1^{-1}, A_2^{-1}, \ldots, \quad A_0 = A.$$

$\rightsquigarrow$ **Idea**: reuse $\widehat{A}_1^{-1}, \widehat{A}_2^{-1}, \ldots$ across different refinement iterations

- Dominant cost: ~~$2\sum_{\ell=0}^{i} k_\ell n^3$~~ $2 \max_{0\le\ell\le i} k_\ell n^3$ flops in precision $u_s$. $k_\ell$ the number of Newton (inner) iterations at $\ell$th refinement step.

- Reduced computational cost & additional storage requirement: particularly interesting when $\widehat{A}_1^{-1}, \widehat{A}_2^{-1}, \ldots$ fits into *fast memory*

**Quantity of interest** in computational cost comparison:

$$k^\sigma =: \sum_{\ell=0}^{i} k_\ell \quad \text{or} \quad k^{\max} =: \max_{0\le\ell\le i} k_\ell.$$

Compare computational costs for mpIR with different solver precisions:

$$k^\sigma_{\text{low}} / k^\sigma_{\text{high}} \quad \text{or} \quad k^{\max}_{\text{low}} / k^{\max}_{\text{high}}$$

# Iterative refinement for Lyapunov equations

## Alternative cost model

- Fact: in each refinement (outer) step, same sequence generated:

$$A_1^{-1}, A_2^{-1}, \ldots, \quad A_0 = A.$$

$\rightsquigarrow$ **Idea**: reuse $\widehat{A}_1^{-1}, \widehat{A}_2^{-1}, \ldots$ across different refinement iterations

- Dominant cost: ~~$2\sum_{\ell=0}^{i} k_\ell n^3$~~ $2 \max_{0\le\ell\le i} k_\ell n^3$ flops in precision $u_s$. $k_\ell$ the number of Newton (inner) iterations at $\ell$th refinement step.

- Reduced computational cost & additional storage requirement: particularly interesting when $\widehat{A}_1^{-1}, \widehat{A}_2^{-1}, \ldots$ fits into *fast memory*

**Quantity of interest** in computational cost comparison:

$$k^\sigma =: \sum_{\ell=0}^{i} k_\ell \quad \text{or} \quad k^{\max} =: \max_{0\le\ell\le i} k_\ell.$$

Compare computational costs for mpIR with different solver precisions:

$$k^\sigma_{\text{low}}/k^\sigma_{\text{high}} \quad \text{or} \quad k^{\max}_{\text{low}}/k^{\max}_{\text{high}}$$

# Numerical experiments
## Experimental setting

- bf16, fp32, fp64 in MATLAB, bf16 simulated by chop (**Higham & Pranesh, '19**)
- We measure, in fp64,

$$\mathrm{Res}(\widehat{Z},\widehat{Y}) = \frac{\|A\widehat{Z}\widehat{Y}\widehat{Z}^T + \widehat{Z}\widehat{Y}\widehat{Z}^T A^T + W\|_F}{\|W\|_F + 2\|\widehat{Z}\widehat{Y}\widehat{Z}^T\|_F\|A\|_F}.$$

### Parameters

- mpIR convergence tol $= nu$
- Max (outer) refinement steps $= 50$
- Max (inner) Newton iterations $= 50$

### Runtime model

$$\rho = \frac{\text{average time of bf16 flop}}{\text{average time of high-precision flop}} = \frac{t_{\mathrm{bf16}}}{t_{\mathrm{high}}}$$

Dominant runtime in bf16 $T_{\mathrm{bf16}} = 2n^3 k_{\mathrm{bf16}} \cdot t_{\mathrm{bf16}}$

Dominant runtime in high $T_{\mathrm{high}} = 2n^3 k_{\mathrm{high}} \cdot t_{\mathrm{high}}$

$$T_{\mathrm{bf16}} < T_{\mathrm{high}} \iff \frac{k_{\mathrm{bf16}}}{k_{\mathrm{high}}} < \frac{1}{\rho}, \quad 0 \le \rho \ll 1$$

# Pseudorandom matrices, high precision = fp32

Problem ID on the x-axis

bf16 mpIR faster for $\rho$ below the markers

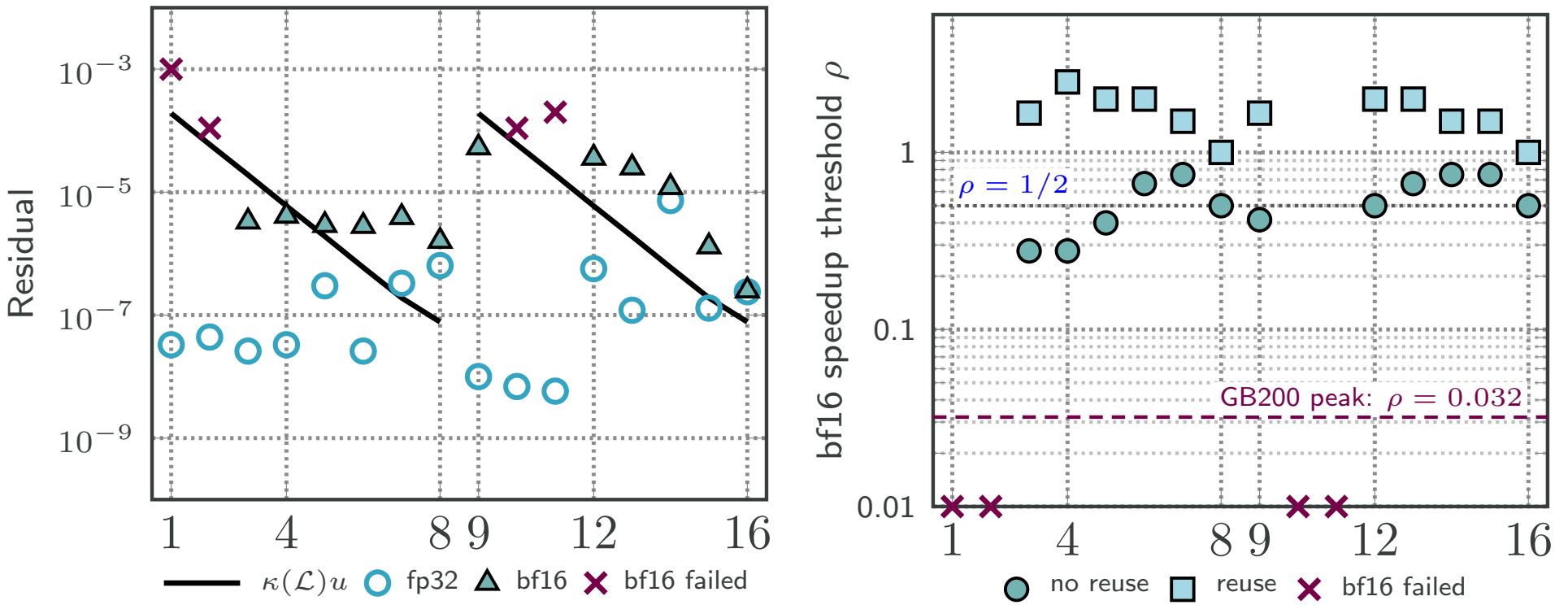



**Figure**: Problems 1–8 have $n = 100$, and problems 9–16 have $n = 1000$.

- Convergence in bf16/fp32 for well-conditioned problems
- GB200 line from dense rates bf16 TC/native fp32 (**NVIDIA, '25**)

# Pseudorandom matrices, high precision = fp32

Problem ID on the x-axis

bf16 mpIR faster for $\rho$ below the markers

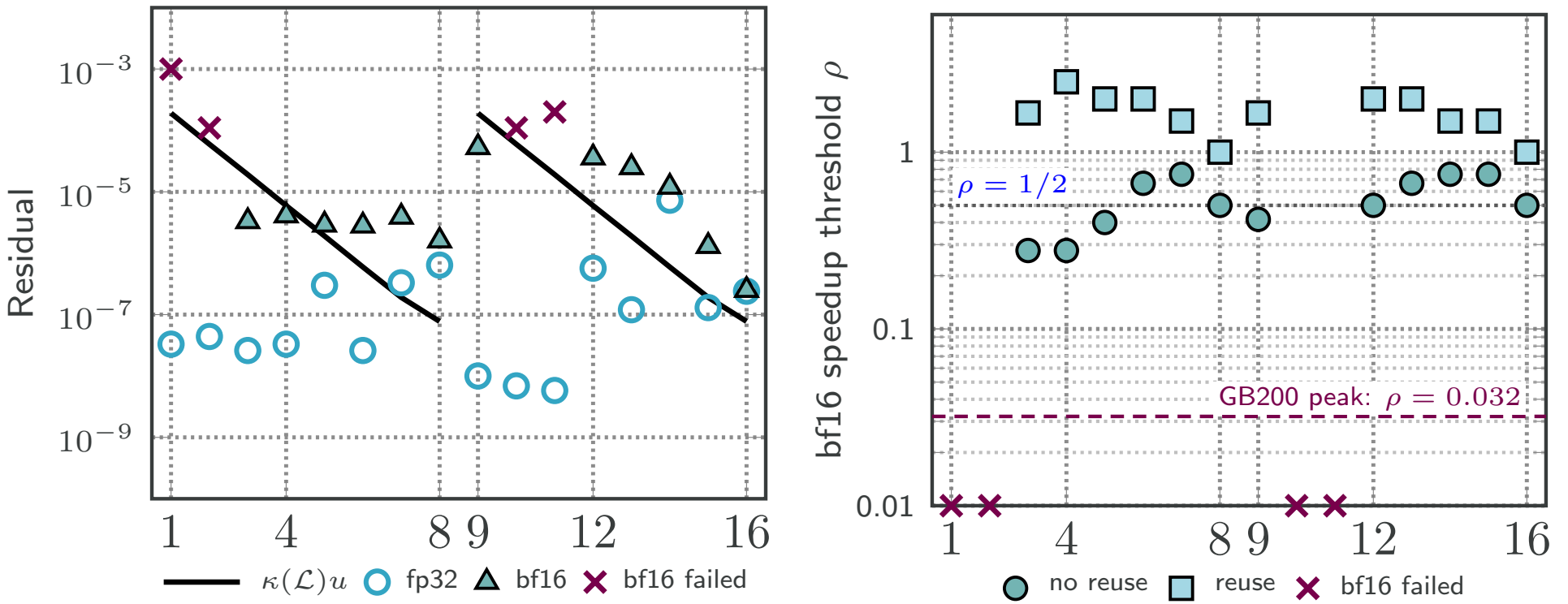
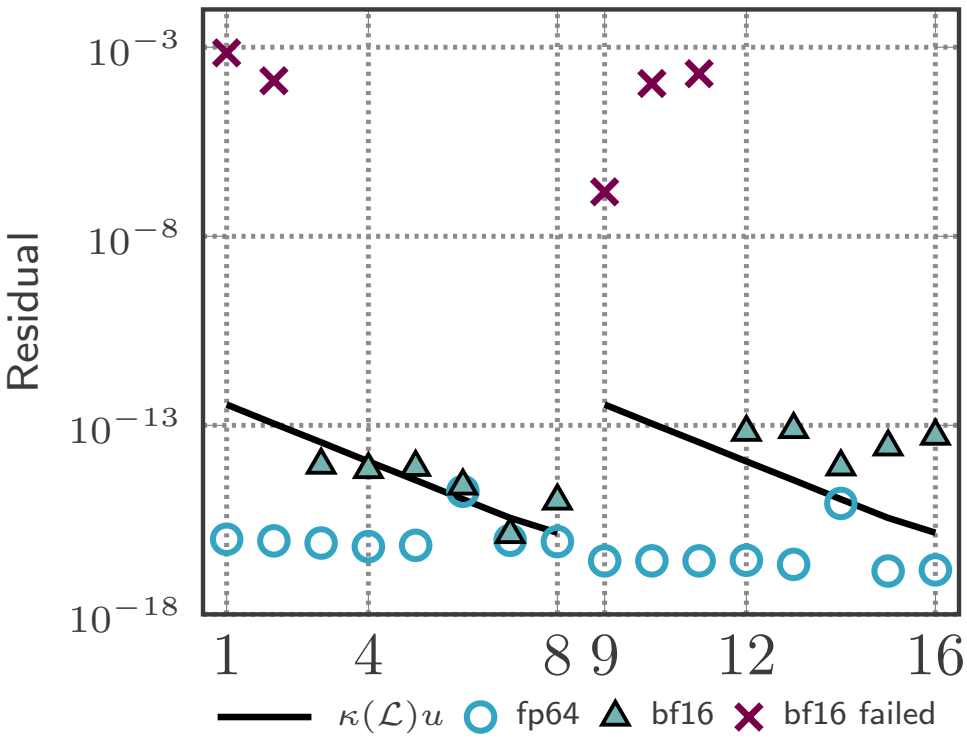



**Figure**: Problems 1–8 have $n = 100$, and problems 9–16 have $n = 1000$.

- Convergence in bf16/fp32 for well-conditioned problems
- GB200 line from dense rates bf16 TC/native fp32 (**NVIDIA, '25**)

# Pseudorandom matrices, high precision = fp64

Problem ID on the x-axis




bf16 mpIR faster for $\rho$ below the markers

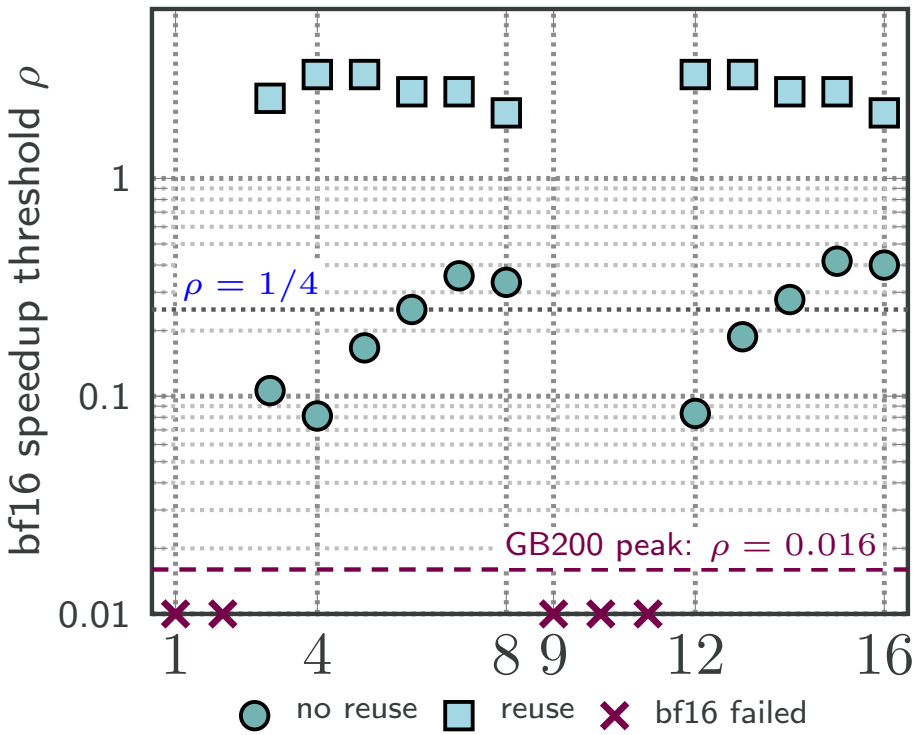
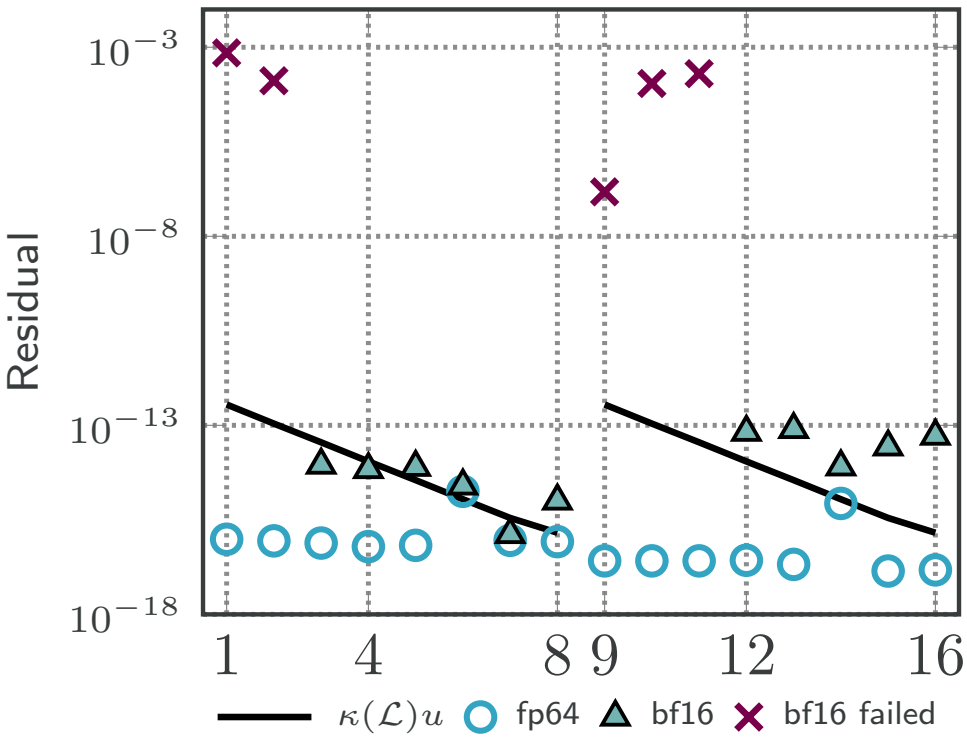



**Figure**: Problems 1–8 have $n = 100$, and problems 9–16 have $n = 1000$.

- Convergence in bf16/fp64 for well-conditioned problems
- GB200 line from dense rates bf16 TC/fp64 TC (**NVIDIA, '25**)

# Pseudorandom matrices, high precision = fp64

Problem ID on the x-axis




bf16 mpIR faster for $\rho$ below the markers

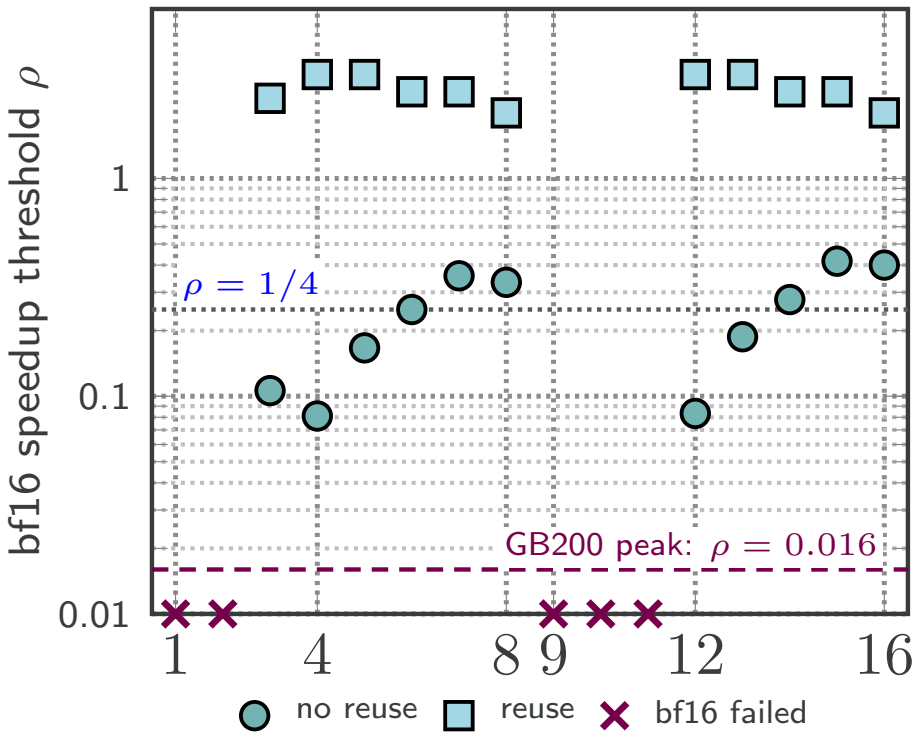



**Figure**: Problems 1–8 have $n = 100$, and problems 9–16 have $n = 1000$.

- Convergence in bf16/fp64 for well-conditioned problems
- GB200 line from dense rates bf16 TC/fp64 TC (**NVIDIA, '25**)

# Outline

1. Brief overview on mixed precision
2. Sylvester matrix equations
3. Low-rank Lyapunov matrix equations
4. **Conclusions**

# Conclusions

- Mixed-precision algorithms for two types of linear matrix equations

Sylvester matrix equations:

- Schur-based $\rightsquigarrow$ preferred for dense & moderate-sized problems
- Iterative refinement for high-precision solution to triangular equation
- Exploitation of easy inversion/re-orthogonalization of low-precision unitary matrix

Low-rank Lyapunov equations:

- Chol/LDL$^T$-type mixed-precision iterative refinement frameworks
- Focused on sign function Newton iteration as solver
- Reduced precisions such as bf16 works best for *well conditioned* problems

- Done, but not covered in the talk: convergence & rounding error analysis

Next?

- High-performance mixed-precision implementations (low-precision Schur?)
- Reduced precision with ADI/Krylov-based solver inside mpIR?

Check the preprints for more details:

arxiv.org/abs/2503.03456 and arxiv.org/abs/2510.02126

Slides available on xiaobo-liu.github.io/talks

**Thank you for your attention.**

# Conclusions

- Mixed-precision algorithms for two types of linear matrix equations

Sylvester matrix equations:

- Schur-based $\rightsquigarrow$ preferred for dense & moderate-sized problems
- Iterative refinement for high-precision solution to triangular equation
- Exploitation of easy inversion/re-orthogonalization of low-precision unitary matrix

Low-rank Lyapunov equations:

- Chol/LDL$^T$-type mixed-precision iterative refinement frameworks
- Focused on sign function Newton iteration as solver
- Reduced precisions such as bf16 works best for *well conditioned* problems

- Done, but not covered in the talk: convergence & rounding error analysis

Next?

- High-performance mixed-precision implementations (low-precision Schur?)
- Reduced precision with ADI/Krylov-based solver inside mpIR?

Check the preprints for more details:

arxiv.org/abs/2503.03456 and arxiv.org/abs/2510.02126

Slides available on xiaobo-liu.github.io/talks

**Thank you for your attention.**

# Conclusions

- Mixed-precision algorithms for two types of linear matrix equations

Sylvester matrix equations:

- Schur-based $\rightsquigarrow$ preferred for dense & moderate-sized problems
- Iterative refinement for high-precision solution to triangular equation
- Exploitation of easy inversion/re-orthogonalization of low-precision unitary matrix

Low-rank Lyapunov equations:

- Chol/LDL$^T$-type mixed-precision iterative refinement frameworks
- Focused on sign function Newton iteration as solver
- Reduced precisions such as bf16 works best for *well conditioned* problems

- Done, but not covered in the talk: convergence & rounding error analysis

Next?

- High-performance mixed-precision implementations (low-precision Schur?)
- Reduced precision with ADI/Krylov-based solver inside mpIR?

Check the preprints for more details:

arxiv.org/abs/2503.03456 and arxiv.org/abs/2510.02126

Slides available on xiaobo-liu.github.io/talks

**Thank you for your attention.**

# Conclusions

- Mixed-precision algorithms for two types of linear matrix equations

Sylvester matrix equations:

- Schur-based $\rightsquigarrow$ preferred for dense & moderate-sized problems
- Iterative refinement for high-precision solution to triangular equation
- Exploitation of easy inversion/re-orthogonalization of low-precision unitary matrix

Low-rank Lyapunov equations:

- Chol/LDL$^T$-type mixed-precision iterative refinement frameworks
- Focused on sign function Newton iteration as solver
- Reduced precisions such as bf16 works best for *well conditioned* problems

- Done, but not covered in the talk: convergence & rounding error analysis

Next?

- High-performance mixed-precision implementations (low-precision Schur?)
- Reduced precision with ADI/Krylov-based solver inside mpIR?

Check the preprints for more details:

arxiv.org/abs/2503.03456 and arxiv.org/abs/2510.02126

Slides available on xiaobo-liu.github.io/talks

**Thank you for your attention.**

# 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.

P. Benner, P. Ezzatti, D. Kressner, E. S. Quintana-Ortí, and A. Remón.
A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU–GPU platforms.
*Parallel Comput.*, 37:439–450, 2011.

P. Benner and E. S. Quintana-Ortí.
Solving stable generalized Lyapunov equations with the matrix sign function.
*Numer. Algorithms*, 20:75–100, 1999.

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.

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

N. J. Higham and S. Pranesh.
Simulating low precision floating-point arithmetic.
*SIAM J. Sci. Comput.*, 41(5):C585–C602, 2019.

N. Komaroff.
Simultaneous eigenvalue lower bounds for the Lyapunov matrix equation.
*IEEE Trans. Automat. Control*, 33(1):126–128, 1988.

V. B. Larin and F. A. Aliev.
Construction of square root factor for solution of the Lyapunov matrix equation.
*Syst. Control Lett.*, 20(2):109–112, 1993.

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

J. D. Roberts.
Linear model reduction and solution of the algebraic Riccati equation by use of the sign function.
*Int. J. Control*, 32(4):677–687, 1980.