From Tensor Rank to the Inversion of a Complex Matrix

Zhen Dai
Lek-Heng Lim
Ke Ye

1

Bilinear Algorithms

- Consider a bilinear operator

$$\beta : \mathbb{U} \times \mathbb{V} \longrightarrow \mathbb{W}$$

- A bilinear algorithm is a decomposition

$$\beta(u, v) = \sum_{i=1}^{r} \phi_i(u)\psi_i(v)w_i$$

2

Growth Factor

Definition 3.1. Let $\mathbb{U}, \mathbb{V}, \mathbb{W}$ be three finite-dimensional real vector spaces. A *decomposition* of a bilinear operator $\beta : \mathbb{U} \times \mathbb{V} \to \mathbb{W}$ is a sequence $\mathcal{D} = (\varphi_i, \psi_i, w_i)_{i=1}^r$ with

$$\beta = \sum_{i=1}^{r} \varphi_i \otimes \psi_i \otimes w_i, \tag{3.2}$$

where $\varphi_i : \mathbb{U} \to \mathbb{R}$ and $\psi_i : \mathbb{V} \to \mathbb{R}$ are linear functionals and $w_i \in \mathbb{W}$, $i = 1, \ldots, r$. An *algorithm* $\widehat{\beta}_{\mathcal{D}}$ given by the decomposition $\mathcal{D}$ takes $(u, v) \in \mathbb{U} \times \mathbb{V}$ as inputs and computes the output $\beta(u, v)$ in three steps:

(i) computes $\varphi_i(u)$ and $\psi_i(v)$, $i = 1, \ldots, r$;
(ii) computes $\varphi_i(u)\psi_i(v)w_i$, $i = 1, \ldots, r$;
(iii) computes $\sum_{i=1}^r \varphi_i(u)\psi_i(v)w_i$.

The *growth factor* of the algorithm $\widehat{\beta}_{\mathcal{D}}$ is defined as

$$\gamma(\widehat{\beta}_{\mathcal{D}}) := \sum_{i=1}^{r} \|\varphi_i \otimes \psi_i \otimes w_i\| = \sum_{i=1}^{r} \|\varphi_i\|_* \|\psi_i\|_* \|w_i\|.$$

3

Tensor Rank

- The bilinear complexity of an algorithm is the number of terms r in the decomposition

$$\beta = \sum_{i=1}^{r} \phi_i \otimes \psi_i \otimes w_i.$$

- The tensor rank corresponds to the optimal speed of evaluating this bilinear operator

$$\operatorname{rank}(\beta) := \min\left\{ r : \beta = \sum_{i=1}^{r} \varphi_i \otimes \psi_i \otimes w_i \right\}$$

4

Example

- (a + ib)(c + id) = (ac - bd) + i(ad + bc)
- Viewed as a bilinear operator $\beta : \mathbb{R}^2 \times \mathbb{R}^2 \longrightarrow \mathbb{R}^2$
- Method 1: compute ac, bd, ad, bc
- Method 2: compute (a+b)(c+d), ac, bd

$$e_1^*(a,b) = a, \quad e_2^*(a,b) = b$$

$$\widehat{\beta}_{\mathsf{R}} = (e_1^* \otimes e_1^* - e_2^* \otimes e_2^*) \otimes e_1 + (e_1^* \otimes e_2^* + e_2^* \otimes e_1^*) \otimes e_2,$$
$$\widehat{\beta}_{\mathsf{G}} = (e_1^* + e_2^*) \otimes (e_1^* + e_2^*) \otimes e_2 + e_1^* \otimes e_1^* \otimes (e_1 - e_2) - e_2^* \otimes e_2^* \otimes (e_1 + e_2),$$

5

Example

- Growth factor of method1 is 4.
- Growth factor of method2 is $2(1+\sqrt{2})$.

$$\widehat{\beta}_{\mathsf{R}} = (e_1^* \otimes e_1^* - e_2^* \otimes e_2^*) \otimes e_1 + (e_1^* \otimes e_2^* + e_2^* \otimes e_1^*) \otimes e_2,$$
$$\widehat{\beta}_{\mathsf{G}} = (e_1^* + e_2^*) \otimes (e_1^* + e_2^*) \otimes e_2 + e_1^* \otimes e_1^* \otimes (e_1 - e_2) - e_2^* \otimes e_2^* \otimes (e_1 + e_2),$$

- Indeed, the second method is less stable than the first one.

6

Tensor Rank of Complex Matrix Multiplication

- Every algorithm that evaluates complex matrix multiplications requires at least three real matrix multiplications. [Winograd 1971]
- Gauss's algorithm uses the least number of real matrix multiplciations.
- Regular algorithm has the smallest growth factor.
- Does there exist an algorithm that is both fast and accurate?

7

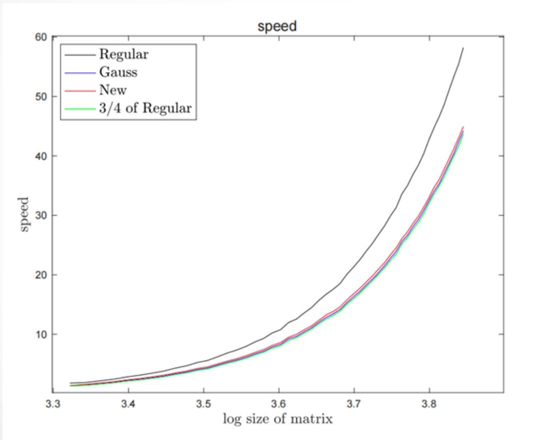
Complex Matrix Multiplication

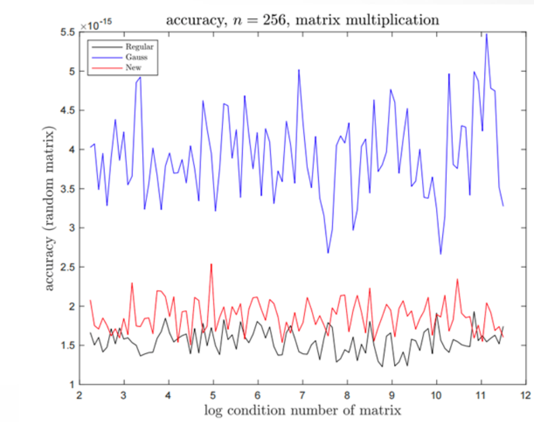
- Uses three real matrix multiplications.
- Has smallest growth factor.

$$
\begin{aligned}
(A+iB)(C+iD) = \frac{1}{2}\Bigg[&\left(A+\frac{1}{\sqrt{3}}B\right)\left(C+\frac{1}{\sqrt{3}}D\right)+\left(A-\frac{1}{\sqrt{3}}B\right)\left(C-\frac{1}{\sqrt{3}}D\right)-\frac{8}{3}BD\Bigg] \\
&+\frac{i\sqrt{3}}{2}\Bigg[\left(A+\frac{1}{\sqrt{3}}B\right)\left(C+\frac{1}{\sqrt{3}}D\right)-\left(A-\frac{1}{\sqrt{3}}B\right)\left(C-\frac{1}{\sqrt{3}}D\right)\Bigg],
\end{aligned}
$$

8

Complex Matrix Multiplication



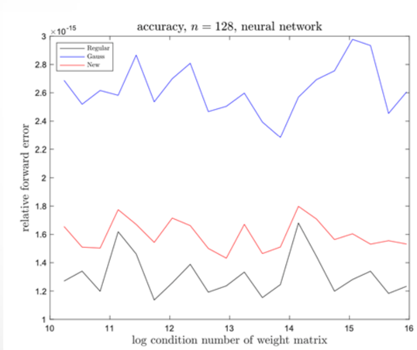


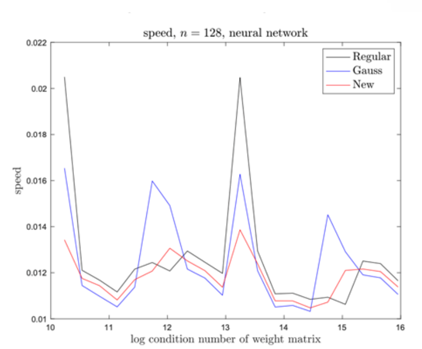
9

Applications

- Complex Neural Network

$$f(W_1, \ldots, W_d, \sigma)(x) := W_d \sigma(W_{d-1} \sigma(\cdots W_2 \sigma(W_1 x) \cdots)).$$





10

Frobenius Inversion

- Z = A+iB

$$Z^{-1} = (A + BA^{-1}B)^{-1} - iA^{-1}B(A + BA^{-1}B)^{-1}$$

11

Optimality

Algorithm 1 Frobenius inversion

Input $A = \mathrm{Re}(Z), B = \mathrm{Im}(Z)$
Output inverse of Z

```
if Z ∈ S_1 then
    set X = A, Y = B;
else if Z ∈ S_2 then
    set X = B, Y = A;
end if
compute X^{-1};
compute X^{-1}Y;
compute YX^{-1}Y;
compute X + YX^{-1}Y;
compute J = (X + YX^{-1}Y)^{-1};
compute K = X^{-1}Y(X + YX^{-1}Y)^{-1};
if Z ∈ S_1 then return Z^{-1} = J − iK;
else if Z ∈ S_2 then return Z^{-1} = K − iJ;
end if
```

Theorem 2.2 (optimality). *Algorithm 1 is optimal in the sense of least number of real matrix multiplications, inversions and additions.*

12

Numerical Properties

Algorithm 2 matrix inversion via LU decomposition

Input $A \in \mathrm{GL}_n(\Bbbk)$
Output inverse of A
1: compute LU factorization of $A = LU$;
2: compute U^{-1};
3: solve for X from $XL = U^{-1}$;
4: **return** X;

Numerical Properties

Theorem 3.1 (threshold). *Let $\mathcal{A}$ be an algorithm for real matrix multiplication. Assume that the running time of $\mathcal{A}$ on pairs of $n \times n$ matrices of which at least one is upper or lower triangular is $\lambda T^{\mathcal{A}}_{\text{mult}}(n)$ for some $0 < \lambda \leq 1$. Then Algorithm 1 is asymptotically faster than Algorithm 2 over $\mathbb{C}$ if and only if $\lim_{n\to\infty} \left(T^{\mathcal{A}}_{\text{inv}}(n)/T^{\mathcal{A}}_{\text{mult}}(n)\right) > 1 + \lambda/2$. In particular, if $\mathcal{A}$ is the usual matrix multiplication algorithm, then Algorithm 1 is asymptotically faster than Algorithm 2 over $\mathbb{C}$ if and only if $\lim_{n\to\infty} \left(T^{\mathcal{A}}_{\text{inv}}(n)/T^{\mathcal{A}}_{\text{mult}}(n)\right) > 5/4$.*

Numerical Properties

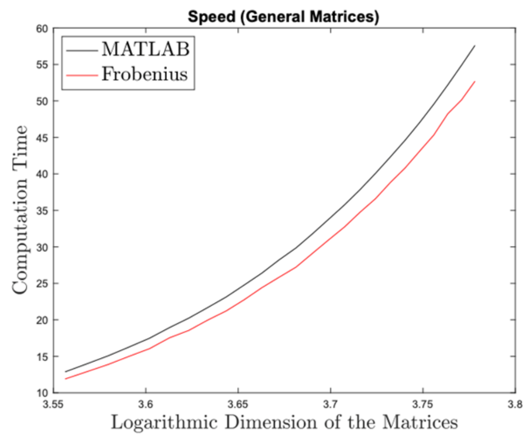


FIGURE 1. comparison of efficiency

15

Numerical Properties

$$\mathrm{res}_L(X,\hat{X}) := \frac{\|\hat{X}X - I\|_{\max}}{\|X\|_{\max}\|\hat{X}\|_{\max}} \quad \text{and} \quad \mathrm{res}_R(X,\hat{X}) := \frac{\|X\hat{X} - I\|_{\max}}{\|X\|_{\max}\|\hat{X}\|_{\max}}$$

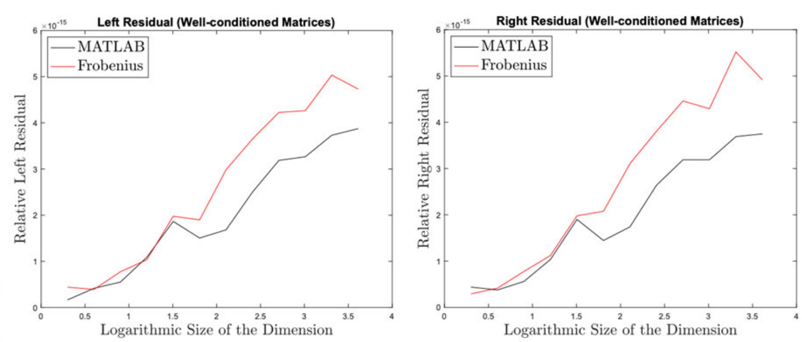


FIGURE 2. comparison of relative residuals

16

Matrix Sign Function

Given a matrix $A \in \mathbb{C}^{n\times n}$, we write $A = ZJZ^{-1}$ where $J = \begin{bmatrix} J_1 & 0 \\ 0 & J_2 \end{bmatrix}$ is the Jordan canonical form of A such that eigenvalues of A in the diagonal of J_1 (resp. J_2) have negative (resp. positive) real parts. The *matrix sign function* is defined to be

$$\mathrm{sign}(A) = Z \begin{bmatrix} -I & 0 \\ 0 & I \end{bmatrix} Z^{-1}.$$

17

Matrix Sign Function

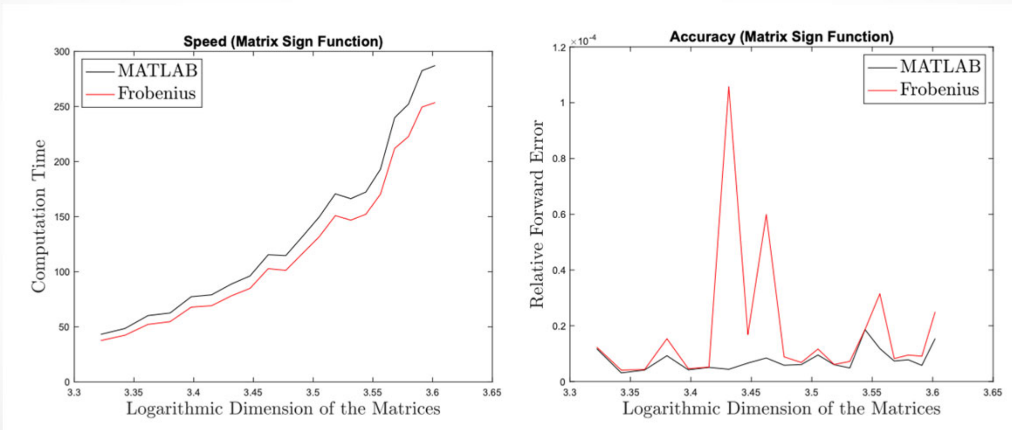
- The matrix sign function of A can be computed via Newton’s method:

$$X_0 = A;$$

$$X_{t+1} = \frac{1}{2}(X_t + X_t^{-1}).$$

18

Matrix Sign Function



19

Sylvester Equation

- Given A,B, and C, want to solve for X in

$$AX + XB = C.$$

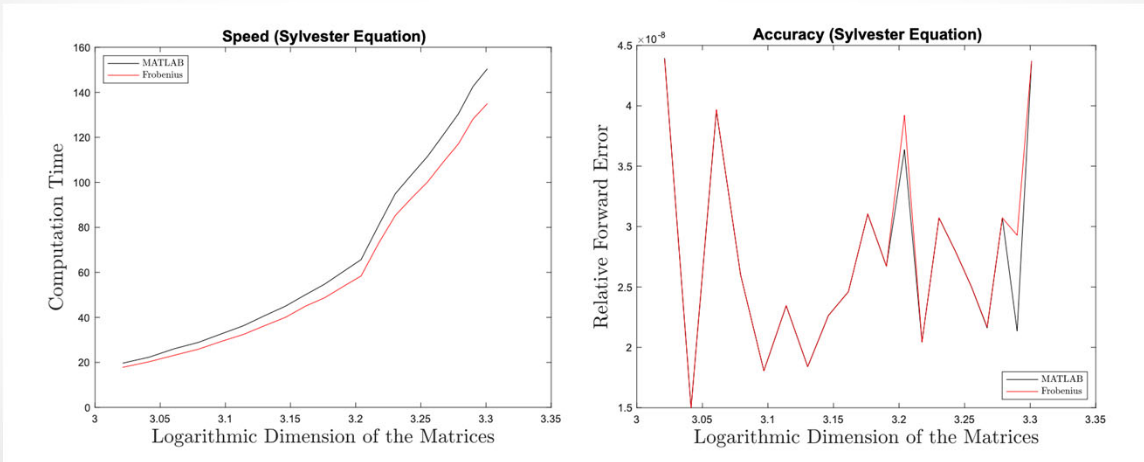
- Can be solved using Newton’s method:

$$X_{t+1} = \frac{1}{2}(X_t + X_t^{-1}),$$

$$X_0 = \begin{bmatrix} A & -C \\ 0 & -B \end{bmatrix}.$$

20

Sylvester Equation



21

Polar Decomposition

- A = UH.
- Compute U using Newton's method:

$$X_0 = A;$$

$$X_{t+1} = \frac{1}{2}(X_t + X_t^{-*}).$$

22

Polar Decomposition

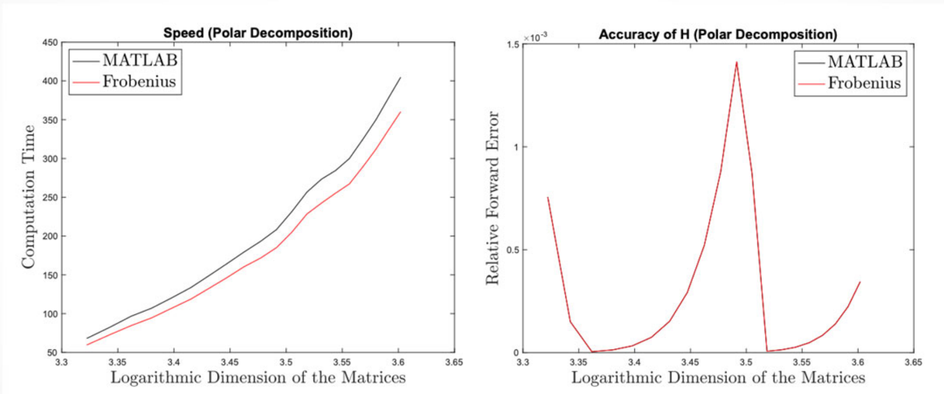


FIGURE 6. comparison on the polar decomposition

23

Thanks for Watching!

24