Fast Hermitian Diagonalization in Near Optimal Precision

Rikhav Shah

UC Berkeley

January 2025

Outline

Motivation

1. SVD is an important tool in the physical and data sciences.
2. $O(n^3)$ and $O(n^\omega)$ algorithms have existed for a long time (Beavers and Denman '73, Francis '61, '62, Wilkinson '68) as has exact arithmetic analysis (Dekker and Traub '71, Hoffmann and Parlett '78)
3. Fast software implementations for floating point arithmetic are widely used (Matlab/Fortran libraries).
   - QR named "top 10 alg" by Dongarra and Sullivan '00
   - Spectral bisection benchmarked by Demmel, Dongarra, Petitet, Robinson, Stanley '97

Motivation

1. SVD is an important tool in the physical and data sciences.
2. $O(n^3)$ and $O(n^\omega)$ algorithms have existed for a long time (Beavers and Denman '73, Francis '61, '62, Wilkinson '68) as has exact arithmetic analysis (Dekker and Traub '71, Hoffmann and Parlett '78)
3. Fast software implementations for floating point arithmetic are widely used (Matlab/Fortran libraries).
   - QR named "top 10 alg" by Dongarra and Sullivan '00
   - Spectral bisection benchmarked by Demmel, Dongarra, Petitet, Robinson, Stanley '97
4. No one has actually determined how many bits of precision one needs in the worst case!
   In fact, until recently, not even an asymtotic bound had appeared (Banks, Garza-Vargas, Kulkarni, Srivastava 2022).

How is this possible?

When doing theory: precise definition of "stability" differs work to work, making reliance on sub-routines tricky. It can be tempting to make "mild" assumptions. (e.g. Nakatsukasa and Higham '13 and dependencies).

When doing applications: one just numerically computes the error of the method on a test suite given a particular precision.

How is this possible?

When doing theory: precise definition of "stability" differs work to work, making reliance on sub-routines tricky. It can be tempting to make "mild" assumptions. (e.g. Nakatsukasa and Higham '13 and dependencies).

When doing applications: one just numerically computes the error of the method on a test suite given a particular precision.

Twofold goals of our worst case analysis:

1. Obtain tight asymptotic bound for how many bits of precision are needed.
2. Determine a concrete, reasonable, number of bits for realistic problem instances.

Approximate diagonalization

Floating point model: numbers are rounded to $\mathbf{u}$ relative error.

$$[D, U] = \texttt{eigh}(A)$$

is *backward stable* to level ε if UDU^* is *exactly* a diagonalization of a nearby matrix, i.e.

$$\|A - UDU^*\| \leq \varepsilon \|A\| \qquad \& \qquad \|I - U^*U\| \leq \varepsilon$$

> *NLA: Given your hardware supports precision* $\mathbf{u}$*, what's the smallest value of* ε *a fast algorithm for diagonalization can achieve?*

Typically one aims for $\varepsilon = \text{poly}(n)\mathbf{u}$

Approximate diagonalization

Floating point model: numbers are rounded to $\mathbf{u}$ relative error.

$$[D, U] = \texttt{eigh}(A)$$

is *backward stable* to level ε if UDU^* is *exactly* a diagonalization of a nearby matrix, i.e.

$$\|A - UDU^*\| \leq \varepsilon \|A\| \qquad \& \qquad \|I - U^*U\| \leq \varepsilon$$

TCS: Given a target error ε, how many bits of precision $\lg(1/\mathbf{u})$ do you need?

Typically one aims for $\lg(1/\mathbf{u}) = \lg(1/\varepsilon) + O(\log n)$

Approximate diagonalization

Floating point model: numbers are rounded to $\mathbf{u}$ relative error.

$$[D, U] = \texttt{eigh}(A)$$

is *backward stable* to level ε if UDU^* is *exactly* a diagonalization of a nearby matrix, i.e.

$$\|A - UDU^*\| \leq \varepsilon \|A\| \qquad \& \qquad \|I - U^*U\| \leq \varepsilon$$

TCS: Given a target error ε, how many bits of precision $\lg(1/\mathbf{u})$ do you need?

Typically one aims for $\lg(1/\mathbf{u}) = \lg(1/\varepsilon) + O(\log n)$

Seminal work of Demmel, Dumitriu, Holtz '07 and Demmel, Dumitriu, Holtz, and Kleinberg '07 show how to do matrix multiplication and QR-decomposition stably in near $O(n^\omega)$ time.

Comparison of bounds

BGVKS'22, Pseudospectral Shattering, the Sign Function, and Diagonalization in Nearly Matrix Multiplication Time.
Upper bound:

$$O\big(\log^4(n/\varepsilon)\log(n)\big) \text{ bits}$$

For $n = 4000$ and $\varepsilon = 10^{-15}$, this is more than 682,916,525,000.

This work
Upper bound:

$$\lg(1/\varepsilon) + O(\log(n) + \log\log(1/\varepsilon)) \text{ bits} \qquad (\approx 92)$$

Lower bound:

$$\lg(1/\varepsilon) + 0.5\lg(n) - 2 \text{ bits} \qquad (\approx 59)$$

Outline

Divide-and-conquer

Recursive block diagonalization (Beavers and Denman '73)

$$
\begin{aligned}
A &= U \begin{bmatrix} A_+ & \\ & A_- \end{bmatrix} U^* \\
&= U \begin{bmatrix} U_+ \begin{bmatrix} A_{++} & \\ & A_{+-} \end{bmatrix} U_+^* & \\ & U_- \begin{bmatrix} A_{-+} & \\ & A_{--} \end{bmatrix} U_-^* \end{bmatrix} U^* \\
&= U \begin{bmatrix} U_+ & \\ & U_- \end{bmatrix} \begin{bmatrix} A_{++} & & A_{+-} & \\ & & & A_{-+} \\ A_{-+} & & A_{--} & \\ & A_{+-} & & A_{--} \end{bmatrix} \begin{bmatrix} U_+ & \\ & U_- \end{bmatrix}^* U^*
\end{aligned}
$$

Matrix sign function

Define the function

$$\operatorname{sign}(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \end{cases}.$$

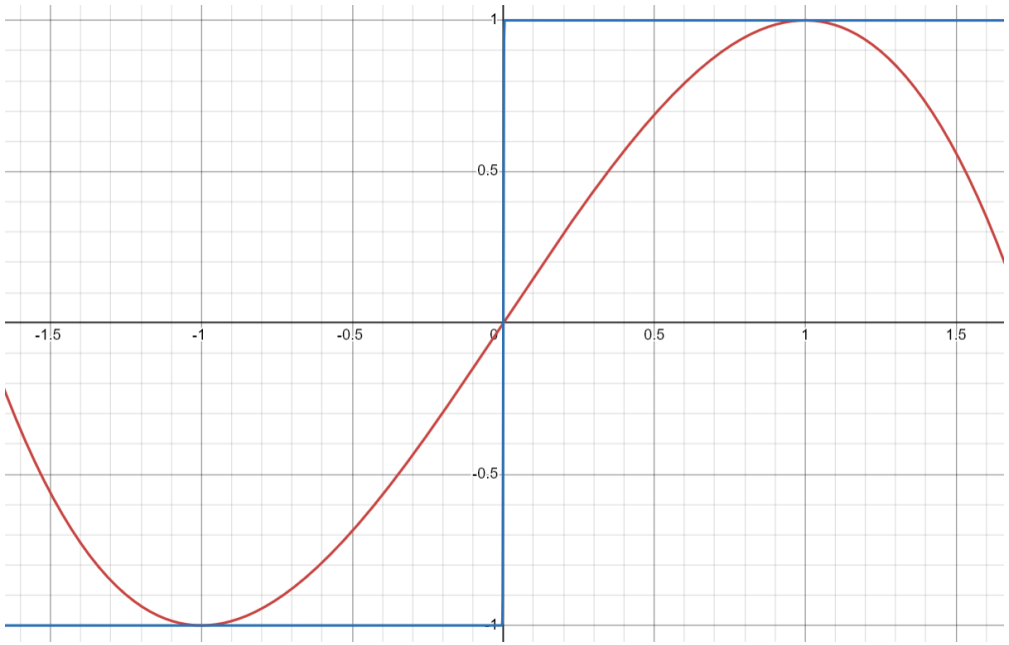
Apply to a Hermitian matrix via the functional calculus:

$$A = \lambda_1 v_1 v_1^* + \cdots + \lambda_n v_n v_n^*$$
$$\operatorname{sign}(A) = \operatorname{sign}(\lambda_1) v_1 v_1^* + \cdots + \operatorname{sign}(\lambda_n) v_n v_n^*$$

Survey of works in Kenney and Laub '95.
Newton: Beavers and Denman '73, Banks et al '22
Newton-Schulz: Bai, Demmel '93, Nakatsukasa, Higham '12.
Weighted versions: Chen, Chow '14, Nakatsukasa, Bai, Gygi '10, Gander '90

Matrix sign: Newton-Schulz iteration

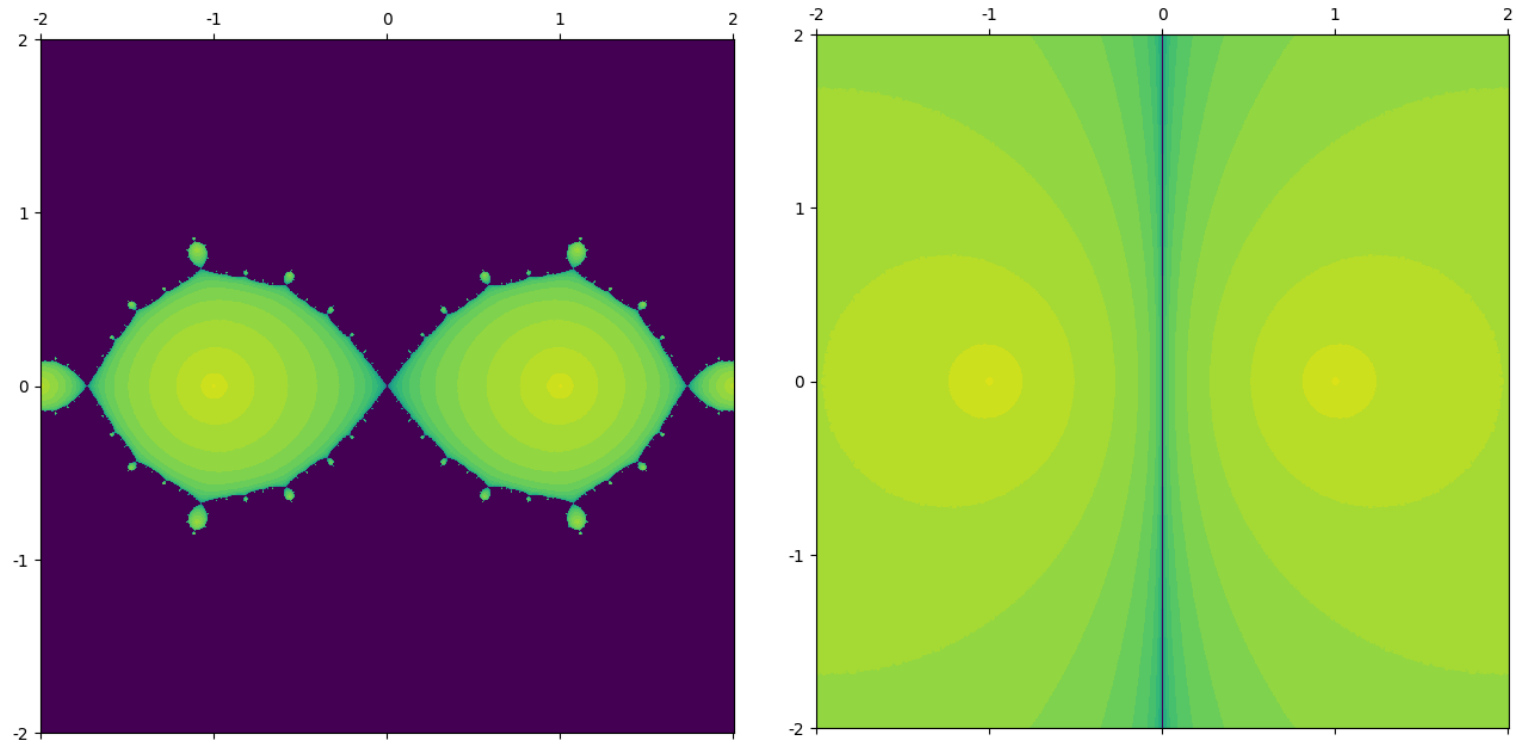
The function $g(x) = \frac{3x - x^3}{2}$ has fixed points at ± 1, and $g'(\pm 1) = 0$.



Plot is $g(x)$ and $g^{(16)}(x)$. Convergence is slow near 0.

Matrix sign: Newton-Schulz iteration

The function $g(x) = \frac{3x - x^3}{2}$ has fixed points at ± 1, and $g'(\pm 1) = 0$.



Convergence plot for $\mathbb{C}$. Left is for Newton-Schulz, right is for Newton $g(z) = \frac{z + z^{-1}}{2}$.

How does computing sign help you?

Note that

$$\begin{aligned}\operatorname{sign}(A) &= \left(\sum_{j:\lambda_j>0} v_j v_j^*\right) - \left(\sum_{j:\lambda_j<0} v_j v_j^*\right) \\ &= P_+ - P_-\end{aligned}$$

where

$$\begin{aligned}P_+ &= \text{projection onto positive eigenspace} \\ P_- &= \text{projection onto negative eigenspace}\end{aligned}$$

If A has no eigenvalue at 0,

$$I = P_+ + P_- \implies P_\pm = \frac{I \pm \operatorname{sign}(A)}{2}$$

Deflation

Given an orthogonal projection matrix P, set

$$V = \texttt{deflate}(P)$$

then the columns of V should be an orthonormal basis for range(P). Set

$$V_\pm = \texttt{deflate}(P_\pm)$$

Then there exists $A_\pm$ such that

$$AV_\pm = V_\pm A_\pm.$$

This gives the desired block diagonalization for $U = \begin{bmatrix} V_+ & V_- \end{bmatrix}$,

$$A = \begin{bmatrix} V_+ & V_- \end{bmatrix} \begin{bmatrix} A_+ & \\ & A_- \end{bmatrix} \begin{bmatrix} V_+^* \\ V_-^* \end{bmatrix} = U \begin{bmatrix} A_+ & \\ & A_- \end{bmatrix} U^*.$$

Deflation

Goal of deflate is to produce a basis V for a projection matrix P.

Algorithm: output the first $\text{rank}(P) = \text{tr}(P)$ columns of the QR decomposition of PG where G is a random matrix.

- RURV: Demmel, Dumitriu, Holtz '07, Demmel, Dumitriu, Rusciano '19
- Deflate: Banks et al. '22
- Single iteration SI: Nakatsukasa and Higham '13

Weakness of existing analysis: if $\hat{V}$ is a true basis and V is the computed basis, then

$$\left\| \hat{V}^* V - I \right\| \leq \varepsilon$$

only implies

$$\left\| \hat{V} - V \right\| = O(\sqrt{\varepsilon}).$$

Recursing

We now have a stable block diagonalization:

$$A = U \begin{bmatrix} A_+ & \\ & A_- \end{bmatrix} U^*.$$

If one computed $\texttt{sign}(A)$ and $\texttt{deflate}(P_\pm)$ correctly, A_+ will contain all the positive eigenvalues of A and A_- the negative eigenvalues. Need to *shift* before recursing.

$$\texttt{eigh}(A_+ - zI) \quad \& \quad \texttt{eigh}(A_- + zI)$$

You make more progress if the split point is near the center of the spectrum.

Need to decrease ε in the recursive calls.

Shifting procedure

To avoid non-convergence of Newton-Schulz iteration: add some randomness (Ballard, Demmel, Dumitriu '11)

Pick recursive split points $\pm\frac{\|A\|}{2}$. So the recursive calls are

$$\texttt{eigh}\left(A_+ - \frac{\|A\|}{2}I\right) \quad \& \quad \texttt{eigh}\left(A_- + \frac{\|A\|}{2}I\right)$$

So the sub-matrices are half the *norm* of the original. Stop when $\|A\| \leq \varepsilon$.

Other ideas:
Shift by the median diagonal entry, this guarantees one eigenvalue on each side (Nakatsukasa and Higham '13).
Use binary search until $\operatorname{tr}\operatorname{sign}(A - zI)$ is small, this guarantees $\Theta(n)$ eigenvalues on each side (Banks et al '22).

Runtime analysis

Cost trade-off: binary search requires many calls to `sign` each iteration but guarantees sub-instances are smaller. This approach requires 1 call to `sign`, but the sub-instances might not be smaller.

Two key ideas in the runtime analysis:

1. The depth is bounded by $\lg(1/\varepsilon)$.
2. The *sum* of the problem sizes within each layer is n. If $n_1 + \cdots + n_k = n$ are the problem sizes, by convexity the cost at each layer is
$$n_1^\omega + \cdots + n_k^\omega \leq n^\omega.$$

Outline

Open problems

1. Efficient and effective shifting strategy? (e.g. analyze the median-of-diagonal method of Nakatsukasa and Higham '13)
2. Bit requirement of general diagonalization? (i.e. improve upon the $O(\log^4(n/\varepsilon)\log(n))$ bound of Banks et al. '22)
3. Analysis of the QR algorithm? (i.e. improve upon or specialize the $O(\log^4(n/\varepsilon)(\log\log(n/\varepsilon)^2))$ bound of Banks et al '22, '23)
4. Do all of this in Lean?