MANCHESTER 1824
The University of Manchester

Stable and Efficient Spectral Divide and Conquer Algorithms for the Symmetric Eigenvalue Decomposition and the SVD

Yuji Nakatsukasa

School of Mathematics

The University of Manchester

yuji.nakatsukasa@manchester.ac.uk

http://www.ma.man.ac.uk/~yuji/

Joint work with Nick Higham

Perspectives on Parallel Numerical Linear Algebra

University of Manchester, July 2012

Symmetric eigendecomposition and SVD

- **Symmetric eigenvalue decomposition** (symeig): for $A = A^T \in \mathbb{R}^{n\times n}$,

$$A = V \cdot \mathrm{diag}(\lambda_1, \ldots, \lambda_n) \cdot V^T \ \left(\equiv V\Lambda V^T\right),$$

where $V^T V = I_n$. $\lambda_1 \geq \cdots \geq \lambda_n$ are the eigenvalues of A.

- **SVD**: for any $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$,

$$A = U \cdot \mathrm{diag}(\sigma_1, \ldots, \sigma_n) \cdot V^T \ \left(\equiv U\Sigma V^T\right),$$

where $U^T U = I_m$, $V^T V = I_n$. $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ are the singular values of A.

Standard algorithm for symmetric eigendecomposition

1. Reduce matrix to tridiagonal form

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \xrightarrow{H_L, H_R} \begin{bmatrix} * & * & & & \\ * & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{bmatrix} \xrightarrow{H_L, H_R} \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & * & * \\ & & * & * & * \\ & & * & * & * \end{bmatrix} \xrightarrow{H_L, H_R} \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & * & \\ & & * & * & * \\ & & & * & * \end{bmatrix} \equiv T$$

   Accumulate orthogonal factors: $A = V_A T V_A^H$ where $V_A = (\prod H_L)^H$
2. Compute eigendecomposition $T = V_B \Lambda V_B^H$ (QR, divide-and conquer, MRRR, ...)
3. Eigendecomposition: $A = (V_A V_B)\Lambda(V_B^H V_A^H) = V \Lambda V^H$

Standard algorithm for symmetric eigendecomposition

1. Reduce matrix to tridiagonal form
   $\Rightarrow$ Expensive in communication cost

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \overset{H_L, H_R}{\rightarrow} \begin{bmatrix} * & * & & & \\ * & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{bmatrix} \overset{H_L, H_R}{\rightarrow} \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & * & * \\ & & * & * & * \\ & & * & * & * \end{bmatrix} \overset{H_L, H_R}{\rightarrow} \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & * & \\ & & * & * & * \\ & & & * & * \end{bmatrix} \equiv T$$

   Accumulate orthogonal factors: $A = V_A T V_A^H$ where $V_A = (\prod H_L)^H$
2. Compute eigendecomposition $T = V_B \Lambda V_B^H$ (QR, divide-and conquer, MRRR, ...)
3. Eigendecomposition: $A = (V_A V_B)\Lambda(V_B^H V_A^H) = V \Lambda V^H$

Standard algorithm for symmetric eigendecomposition

1. Reduce matrix to tridiagonal form
   $\Rightarrow$ Expensive in communication cost

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \overset{H_L, H_R}{\rightarrow} \begin{bmatrix} * & * & & & \\ * & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{bmatrix} \overset{H_L, H_R}{\rightarrow} \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & * & * \\ & & * & * & * \\ & & * & * & * \end{bmatrix} \overset{H_L, H_R}{\rightarrow} \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & * & \\ & & * & * & * \\ & & & * & * \end{bmatrix} \equiv T$$

   Accumulate orthogonal factors: $A = V_A T V_A^H$ where $V_A = (\prod H_L)^H$
2. Compute eigendecomposition $T = V_B \Lambda V_B^H$ (QR, divide-and conquer, MRRR, ...)
3. Eigendecomposition: $A = (V_A V_B)\Lambda(V_B^H V_A^H) = V \Lambda V^H$

We develop completely different algorithms based on

spectral divide-and-conquer

Spectral divide-and-conquer algorithm for symeig

$$A = \begin{bmatrix} * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \end{bmatrix}$$

Spectral divide-and-conquer algorithm for symeig

$$A = \begin{bmatrix} * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \end{bmatrix} \rightarrow V_1^* A V_1 = \begin{bmatrix} * & * & * & * & & \\ * & * & * & * & & \\ * & * & * & * & & \\ * & * & * & * & & \\ & & & & * & * & * & * \\ & & & & * & * & * & * \\ & & & & * & * & * & * \\ & & & & * & * & * & * \end{bmatrix}$$

Spectral divide-and-conquer algorithm for symeig

$$A = \begin{bmatrix} * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \end{bmatrix} \rightarrow V_1^* A V_1 = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ & & & \\ & & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} V_{21} & V_{22} \end{bmatrix}^* V_1^* A V_1 \begin{bmatrix} V_{21} & V_{22} \end{bmatrix} = \begin{bmatrix} * & * \\ * & * & & & & & & \\ & & * & * & & & & \\ & & * & * & & & & \\ & & & & * & * & & \\ & & & & * & * & & \\ & & & & & & * & * \\ & & & & & & * & * \end{bmatrix}$$

Spectral divide-and-conquer algorithm for symeig

$$A = \begin{bmatrix} * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \end{bmatrix} \rightarrow V_1^* A V_1 = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} V_{21} & V_{22} \end{bmatrix}^* V_1^* A V_1 \begin{bmatrix} V_{21} & V_{22} \end{bmatrix} = \begin{bmatrix} * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * \\ * & * \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} V_{31} & & & \\ & V_{32} & & \\ & & V_{33} & \\ & & & V_{34} \end{bmatrix}^* \begin{bmatrix} V_{21} & V_{22} \end{bmatrix}^* V_1^* A V_1 \begin{bmatrix} V_{21} & V_{22} \end{bmatrix} \begin{bmatrix} V_{31} & & & \\ & V_{32} & & \\ & & V_{33} & \\ & & & V_{34} \end{bmatrix} = \mathrm{diag}(\lambda_i)$$

Spectral divide-and-conquer algorithm for symeig

$$A = \begin{bmatrix} * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \end{bmatrix} \rightarrow V_1^* A V_1 = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} V_{21} & \\ & V_{22} \end{bmatrix}^* V_1^* A V_1 \begin{bmatrix} V_{21} & \\ & V_{22} \end{bmatrix} = \begin{bmatrix} * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * \\ * & * \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} V_{31} & & & \\ & V_{32} & & \\ & & V_{33} & \\ & & & V_{34} \end{bmatrix}^* \begin{bmatrix} V_{21} & \\ & V_{22} \end{bmatrix}^* V_1^* A V_1 \begin{bmatrix} V_{21} & \\ & V_{22} \end{bmatrix} \begin{bmatrix} V_{31} & & & \\ & V_{32} & & \\ & & V_{33} & \\ & & & V_{34} \end{bmatrix} = \text{diag}(\lambda_i)$$

– not divide-and-conquer for symmetric tridiagonal eigenproblem!

Executing spectral divide-and-conquer

To obtain

$$[V_+ \; V_-]^* A [V_+ \; V_-] = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix},$$

- V_+, V_- each spans an invariant subspace of A corresponding to a subset of eigenvalues.

Executing spectral divide-and-conquer

To obtain

$$[V_+\ V_-]^* A [V_+\ V_-] = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix},$$

- V_+, V_- each spans an invariant subspace of A corresponding to a subset of eigenvalues.
- A natural idea: let V_+ and V_- span the positive and negative eigenspaces.

  $\rightarrow [V_+\ V_-] \begin{bmatrix} I & \\ & -I \end{bmatrix} [V_+\ V_-]^T = \text{sign}(A)$, the matrix sign function of A. Since $A = A^T$, $\text{sign}(A)$ is equivalent to U, the unitary polar factor in the polar decomposition.

Polar decomposition

For any $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$,

$$A = UH,$$

where $U^T U = I_n$, H: symmetric positive semidefinite

- U is called the unitary polar factor of A.
- Connection with SVD: $A = U_* \Sigma V_*^T = (U_* V_*^T) \cdot (V_* \Sigma V_*^T) = UH$.
- Can be used as first step for computing the SVD.

–[Higham and Papadimitriou (1993)]

polar/eigen decompositions of symmetric matrices

The polar/eigen decompositions of symmetric A:

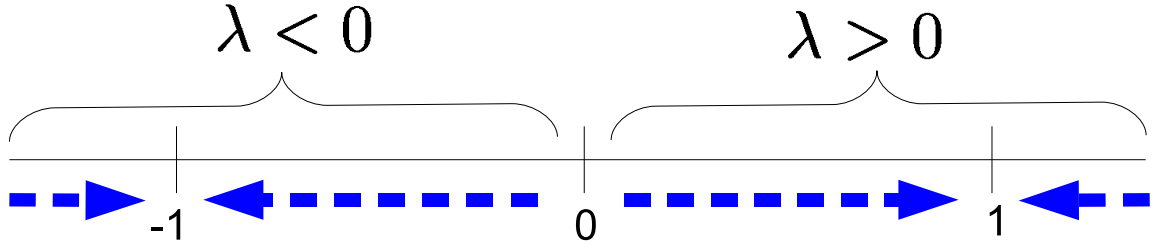
$$A = UH$$

$$= [V_+ \; V_-] \begin{bmatrix} \Lambda_+ & \\ & \Lambda_- \end{bmatrix} [V_+ \; V_-]^T$$

where $\mathrm{diag}(\Lambda_-) < 0$, are related by $U = [V_+ \; V_-] \begin{bmatrix} I & \\ & -I \end{bmatrix} [V_+ \; V_-]^T$:

$$UH = \left([V_+ \; V_-] \begin{bmatrix} I & \\ & -I \end{bmatrix} [V_+ \; V_-]^T \right) \cdot \left([V_+ \; V_-] \begin{bmatrix} \Lambda_+ & \\ & |\Lambda_-| \end{bmatrix} [V_+ \; V_-]^T \right)$$

- $U = f(A)$, where $f(\lambda)$ maps $\lambda > 0$ to 1, $\lambda < 0$ to -1



QR-based iteration for polar decomposition

DWH (Dynamically Weighted Halley): –[N., Bai, Gygi, SIMAX 2010]

$$X_{k+1} = X_k(a_k I + X_k^T X_k)(b_k I + c_k X_k^T X_k)^{-1}$$
$$= \frac{b_k}{c_k} X_k + \left(a_k - \frac{b_k}{c_k}\right) X_k (I + c_k X_k^T X_k)^{-1}, \quad X_0 = A/\|A\|_2$$

– $\lim_{k\to\infty} X_k = U$ in $A = UH$

– a_k, b_k, c_k chosen s.t. $\frac{x_k(a_k+x_k^2)}{b_k+c_k x_k^2}$ is the best $(3,2)$-type rational approximation to sign(x) on $[\sigma_{\min}(X_k), \sigma_{\max}(X_k)]$

QR-based iteration for polar decomposition

DWH (Dynamically Weighted Halley): –[N., Bai, Gygi, SIMAX 2010]

$$X_{k+1} = X_k(a_k I + X_k^T X_k)(b_k I + c_k X_k^T X_k)^{-1}$$
$$= \frac{b_k}{c_k} X_k + \left(a_k - \frac{b_k}{c_k}\right) X_k (I + c_k X_k^T X_k)^{-1}, \quad X_0 = A/\|A\|_2$$

– $\lim_{k\to\infty} X_k = U$ in $A = UH$

– a_k, b_k, c_k chosen s.t. $\frac{x_k(a_k+x_k^2)}{b_k+c_k x_k^2}$ is the best $(3,2)$-type rational approximation to sign(x) on $[\sigma_{\min}(X_k), \sigma_{\max}(X_k)]$

- Lemma:

$$Q_1 Q_2^T = X(I + X^T X)^{-1}, \text{ where } \begin{bmatrix} X \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \mathrm{R}$$

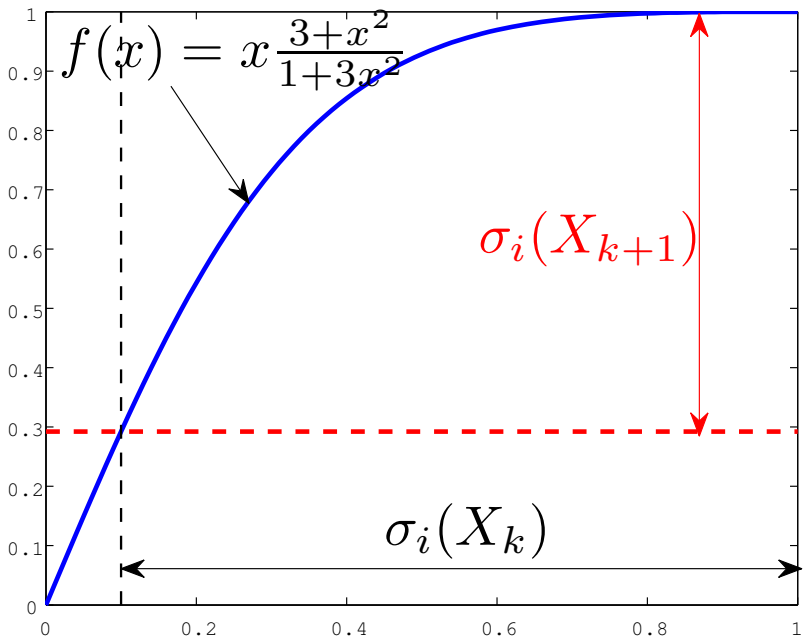
- QR-based DWH (QDWH)

$$\begin{cases} \begin{bmatrix} \sqrt{c_k} X_k \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \\ X_{k+1} = \frac{b_k}{c_k} X_k + \frac{1}{\sqrt{c_k}} \left(a_k - \frac{b_k}{c_k}\right) Q_1 Q_2^T \end{cases}$$

Halley vs. Dynamically weighted Halley

$$X_{k+1} = \begin{cases} X_k(a_k I + X_k^T X_k)(b_k I + c_k X_k^T X_k)^{-1} & \text{(Dynamical Halley)} \\ X_k(3I + X_k^T X_k)(I + 3X_k^T X_k)^{-1} & \text{(Halley)} \end{cases}$$

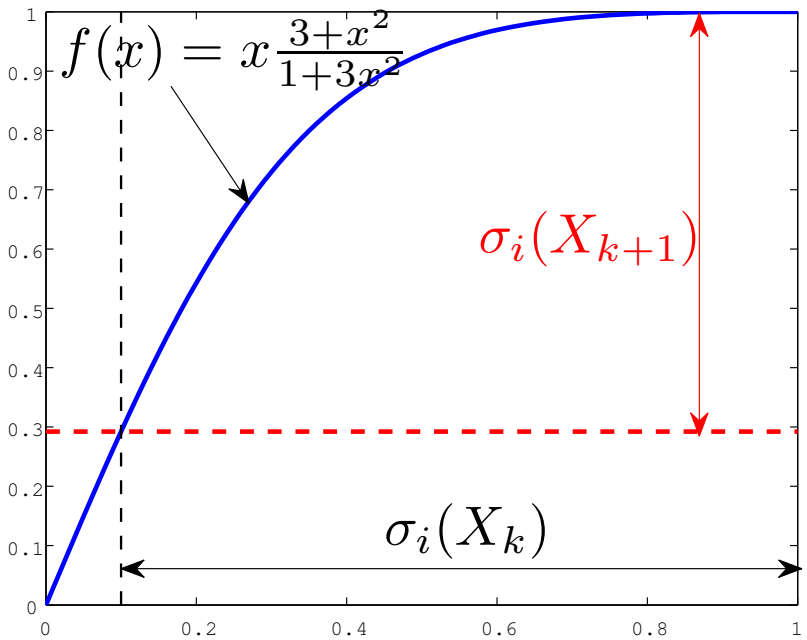
Halley



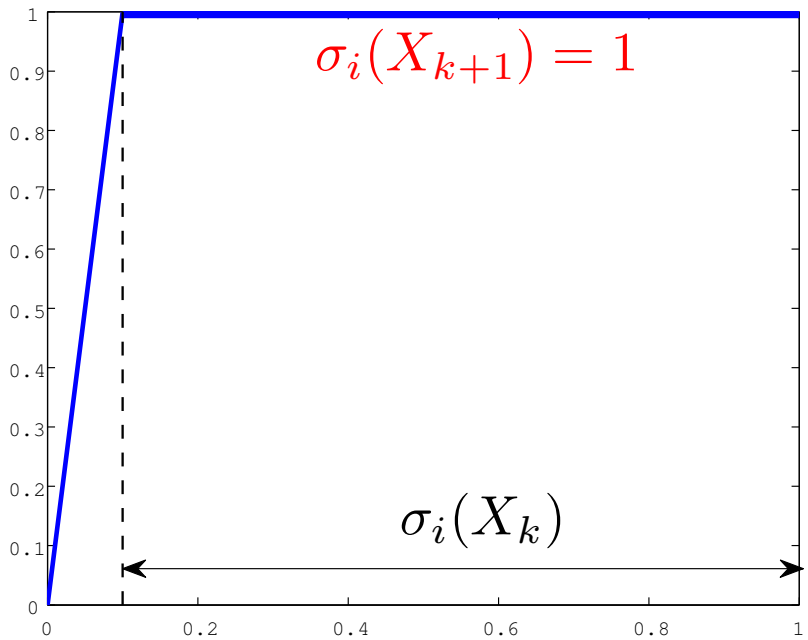
Halley vs. Dynamically weighted Halley

$$X_{k+1} = \begin{cases} X_k(a_k I + X_k^T X_k)(b_k I + c_k X_k^T X_k)^{-1} & \text{(Dynamical Halley)} \\ X_k(3I + X_k^T X_k)(I + 3X_k^T X_k)^{-1} & \text{(Halley)} \end{cases}$$

Halley



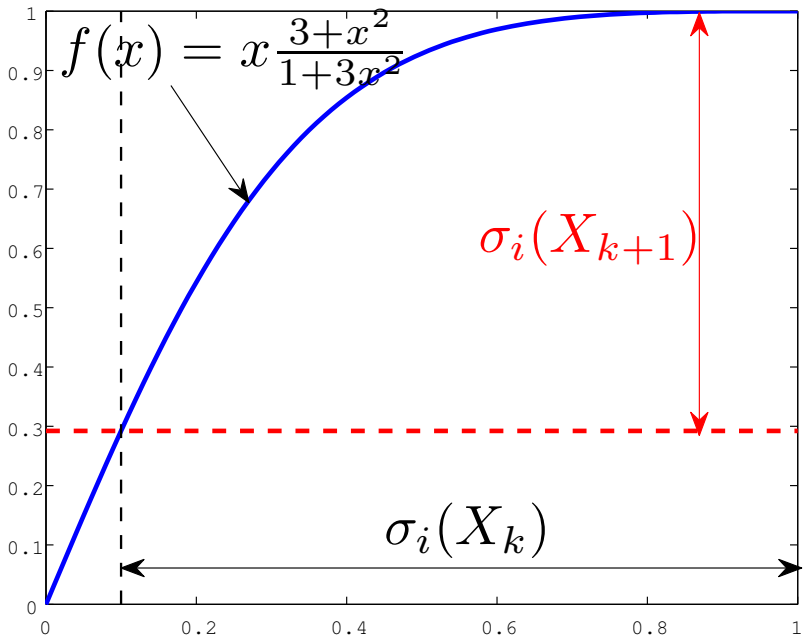
”Optimal” function



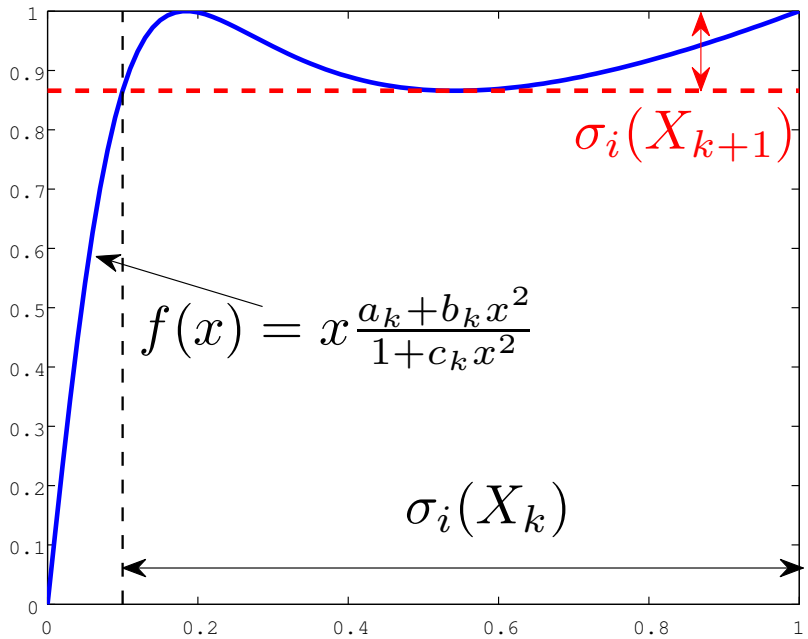
Halley vs. Dynamically weighted Halley

$$X_{k+1} = \begin{cases} X_k(a_k I + X_k^T X_k)(b_k I + c_k X_k^T X_k)^{-1} & \text{(Dynamical Halley)} \\ X_k(3I + X_k^T X_k)(I + 3X_k^T X_k)^{-1} & \text{(Halley)} \end{cases}$$

Halley



Dynamical weighting



- Dynamical weighting uses the best type-(3,2) rational approximation to sign(x), brings $\sigma_i(X_{k+1})$ much closer to 1, thereby speeding up convergence

Algorithm for symmetric eigendecomposition

Algorithm **QDWH-eig**

1. compute polar decomposition $A - \sigma I = UH$
2. compute unitary $[V_+ \; V_-]$ s.t. $(U + I)/2 = V_+ V_+^T$
3. compute $[V_+ \; V_-]^T A [V_+ \; V_-] = \begin{bmatrix} A_+ & E^T \\ E & A_- \end{bmatrix}$
4. repeat steps 1–3 with $A := A_+, A_-$

- Ideal choice of σ: median of eig$(A) \Rightarrow$ dim$(A_+) \simeq$ dim(A_-)
- Dominant cost is in step 1, computing the polar decomposition
  - QDWH needs six or fewer iterations (QR + matrix multiply)

Backward stability

Theorem 1

If subspace iteration is successful and the polar decomposition $A \simeq \widehat{U}\widehat{H}$ *is computed backward stably s.t.*

$$A = \widehat{U}\widehat{H} + \epsilon_1 \|A\|_2, \quad \widehat{U}^T\widehat{U} - I = \epsilon_2, \tag{1}$$

then $A - \widehat{V}\widehat{\Lambda}\widehat{V}^T = \epsilon\|A\|_2$*, that is, QDWH-eig is backward stable.*

Backward stability

Theorem 1

If subspace iteration is successful and the polar decomposition $A \simeq \widehat{U}\widehat{H}$ *is computed backward stably s.t.*

$$A = \widehat{U}\widehat{H} + \epsilon_1 \|A\|_2, \quad \widehat{U}^T\widehat{U} - I = \epsilon_2, \tag{1}$$

then $A - \widehat{V}\widehat{\Lambda}\widehat{V}^T = \epsilon\|A\|_2$*, that is, QDWH-eig is backward stable.*

Theorem 2

(1) *is satisfied if QR factorizations are computed with pivoting in QDWH (in practice even without pivoting).*

–[Nakatsukasa and Higham, SIMAX 2012]

Previous spectral d-c: Implicit Repeated Squaring (IRS)

Algorithm **IRS** —Ballard, Demmel, Dumitriu [2010,LAWN 237]

1. $A_0 = A, B_0 = I$
2. for $j = 0, 1, \ldots,$

$$\begin{bmatrix} B_j \\ -A_j \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} R_j \\ 0 \end{bmatrix},$$

$$A_{j+1} = Q_{12}^T A_j,$$

$$B_{j+1} = Q_{22}^T B_j$$

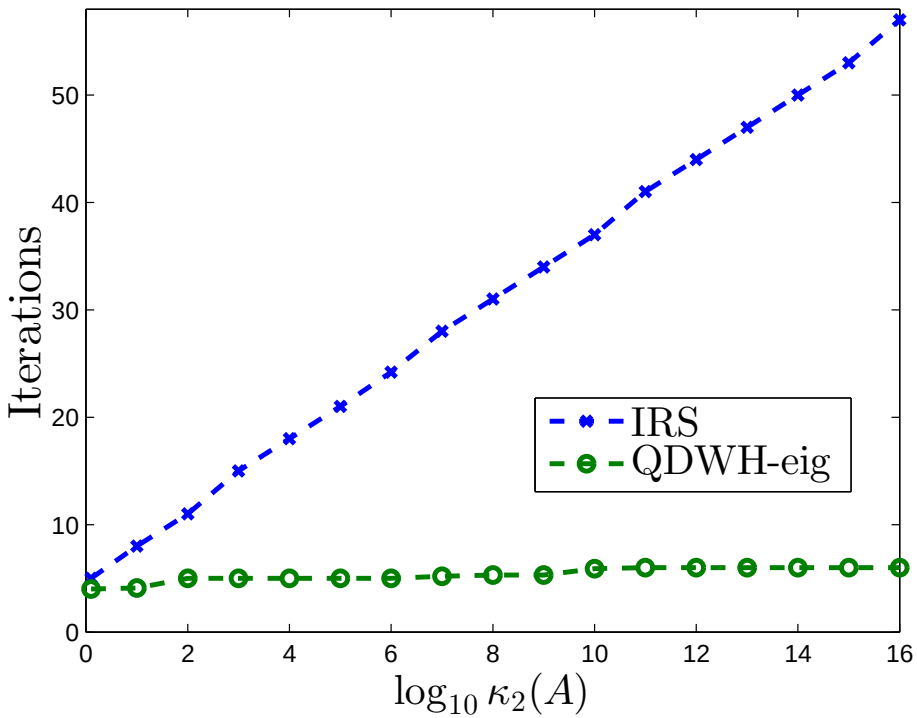
3. rank-revealing QR decomposition $(A_j + B_j)^{-1} A_j = QR$
4. form $Q^T A Q = \begin{bmatrix} A_{11} & E^T \\ E & A_{22} \end{bmatrix}$, confirm $\|E\|_F \lesssim \epsilon \|A\|_F$, repeat 1-4 with $A := A_{11}, A_{22}$

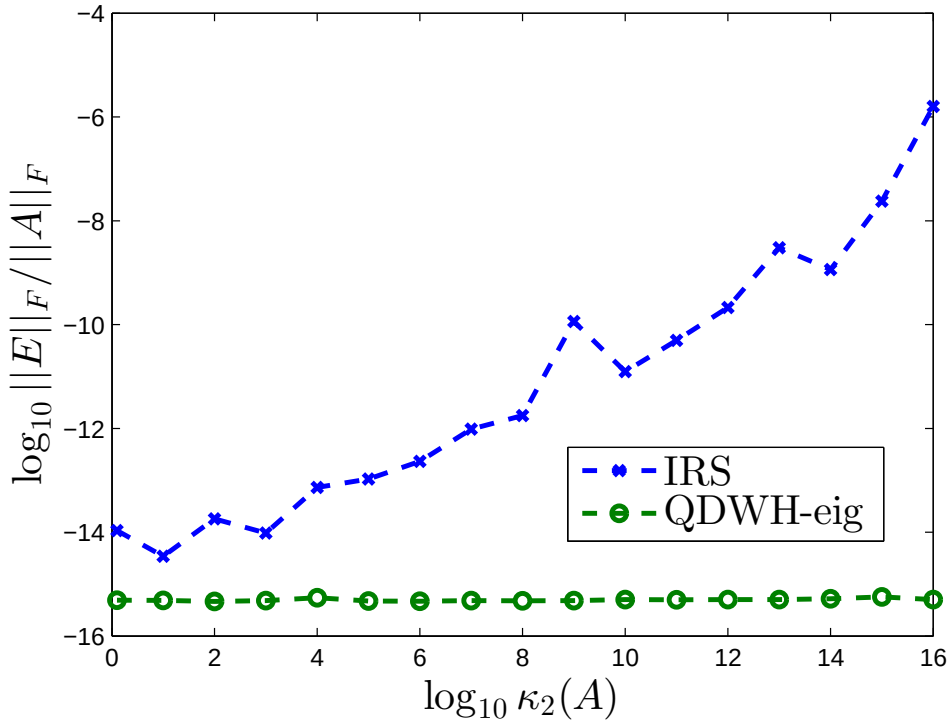
- Motivated by design of **communication-minimizing** algorithms. (QR, matmul, Cholesky are "good" operations)
- $(A_j + B_j)^{-1} A_j = (I + A^{-2^j})^{-1}$: maps $|\lambda| < 1$ to 0, $|\lambda| > 1$ to 1.
- SVD can be computed via eigendecomposition of $\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$.

Spectral d-c algorithms: QDWH vs. IRS

100-by-100 MATLAB randsvd matrices,

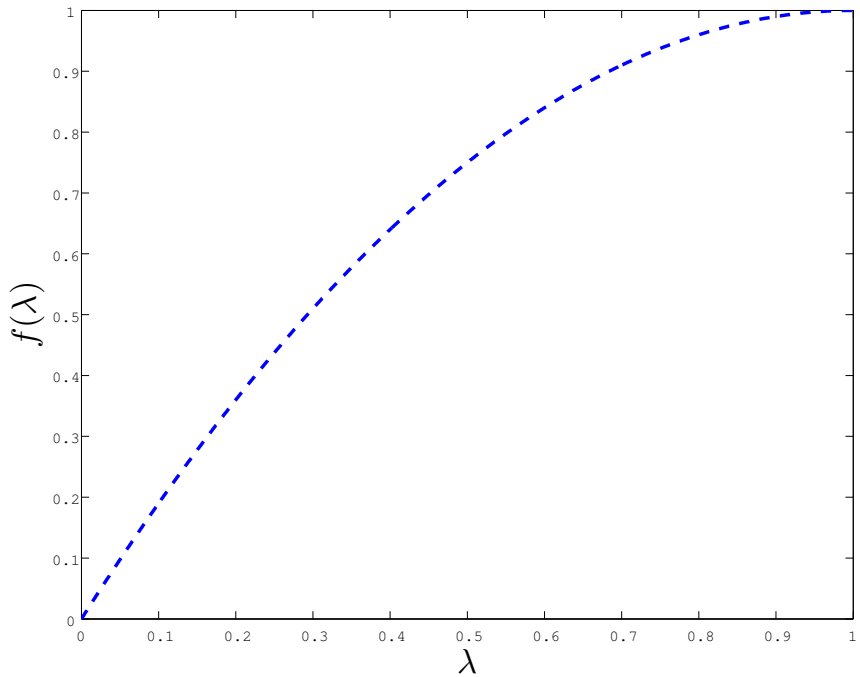
spectral d-c splitting at $\lambda = 0$



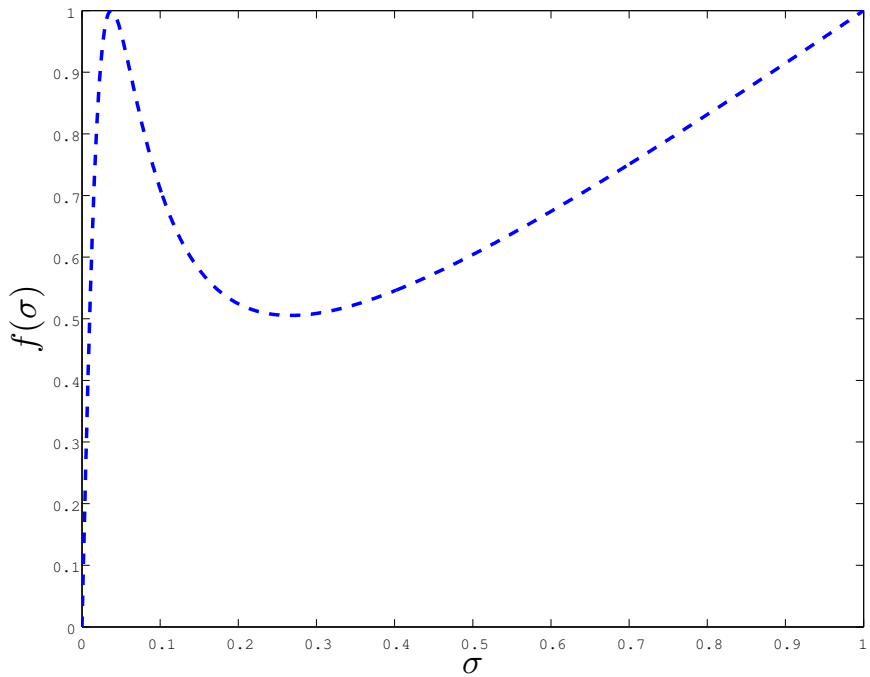


IRS vs. QDWH convergence

IRS: static parameters



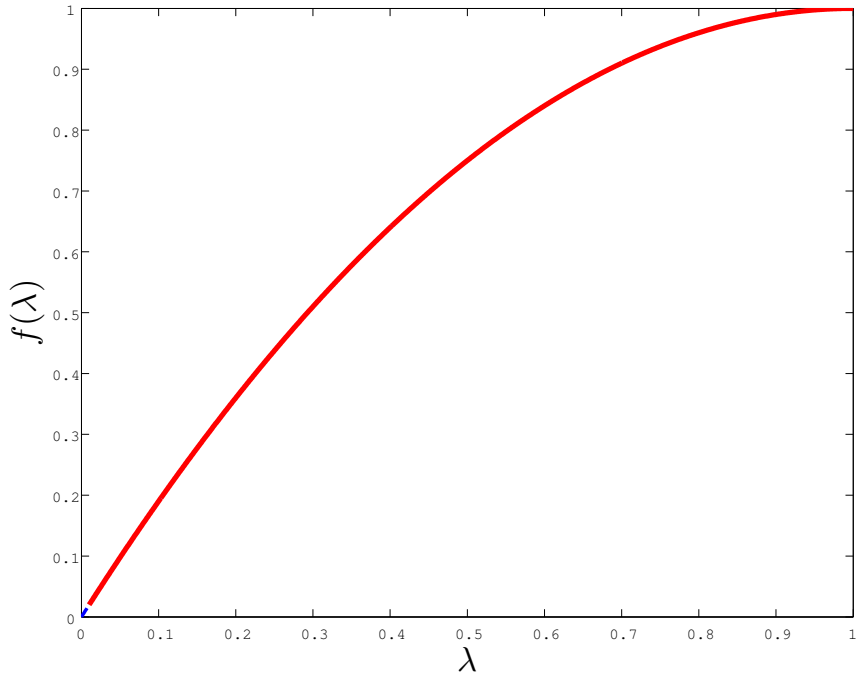
QDWH: dynamical parameters



IRS vs. QDWH convergence

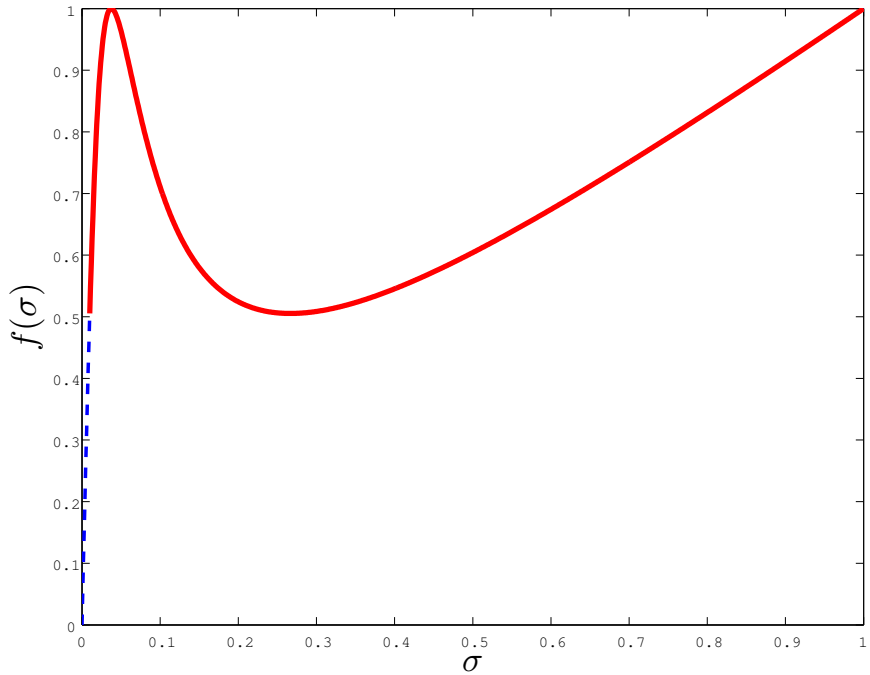
IRS: static parameters

it=0



QDWH: dynamical parameters

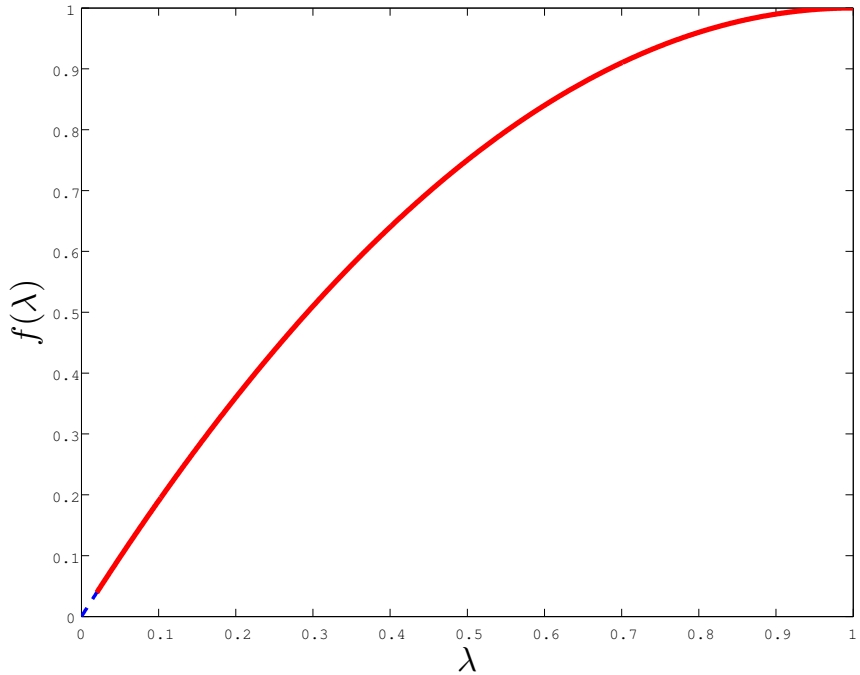
it=0



IRS vs. QDWH convergence

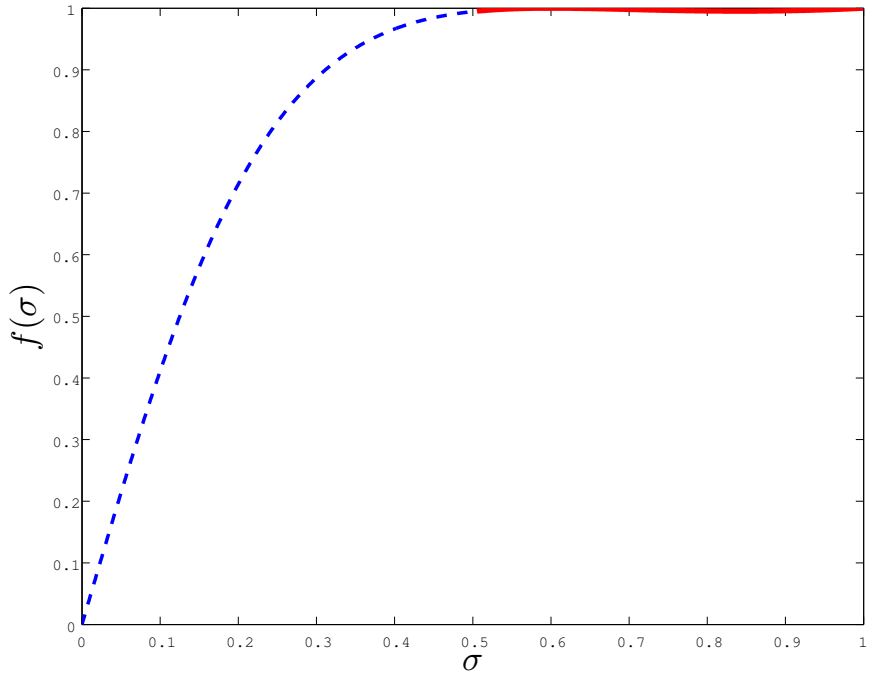
IRS: static parameters

it=1



QDWH: dynamical parameters

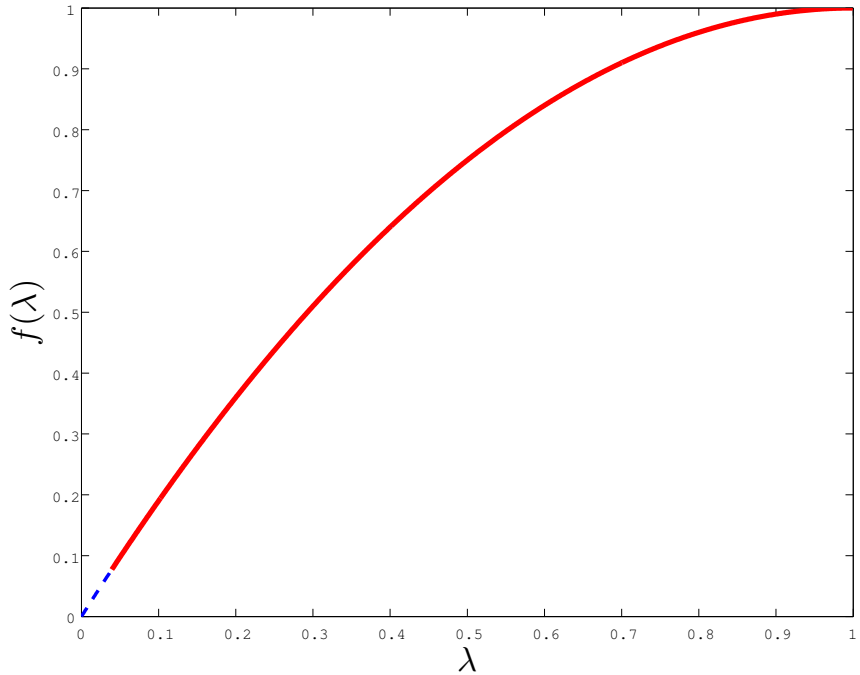
it=1



IRS vs. QDWH convergence

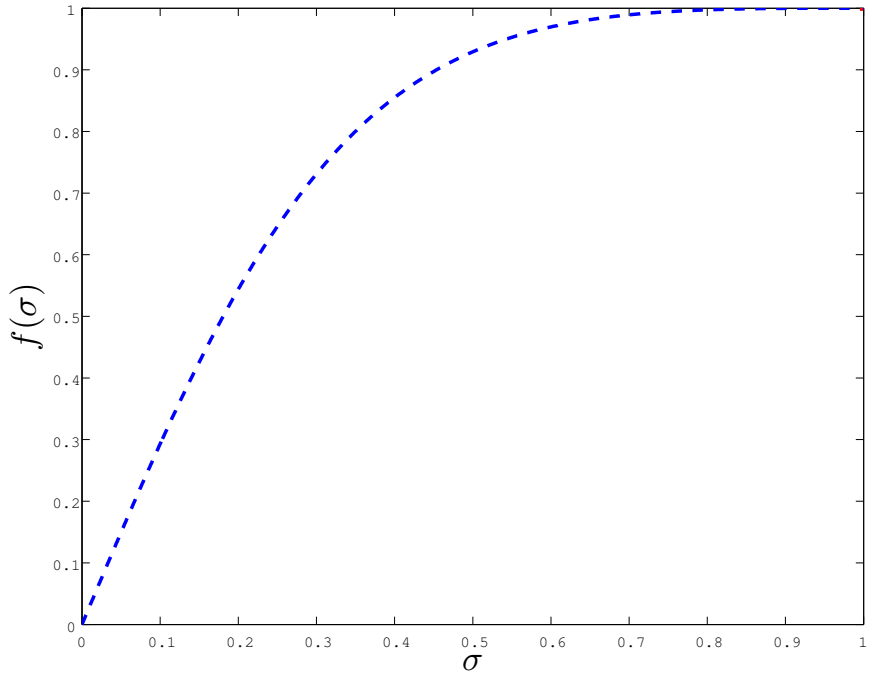
IRS: static parameters

it=2



QDWH: dynamical parameters

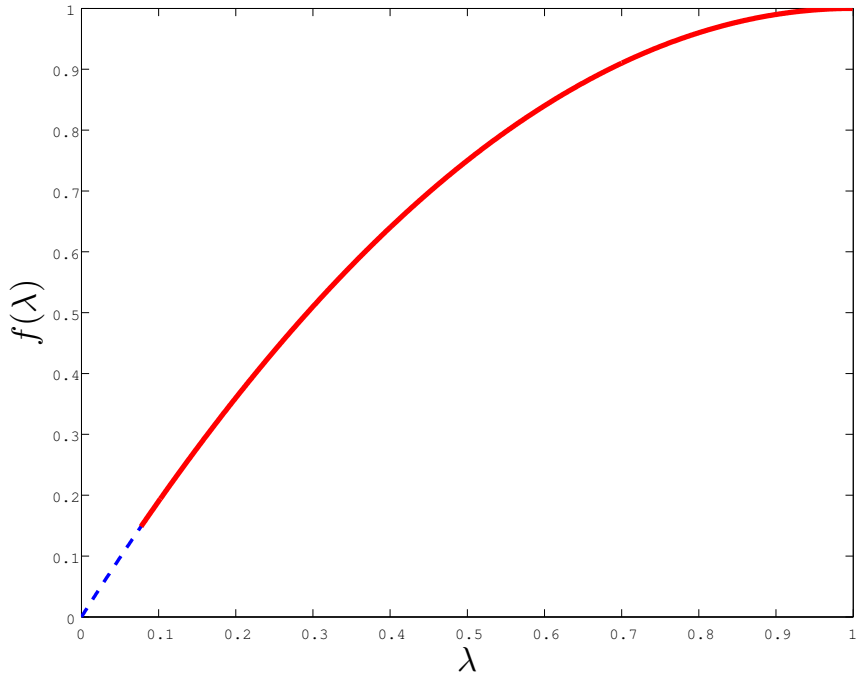
it=2



IRS vs. QDWH convergence

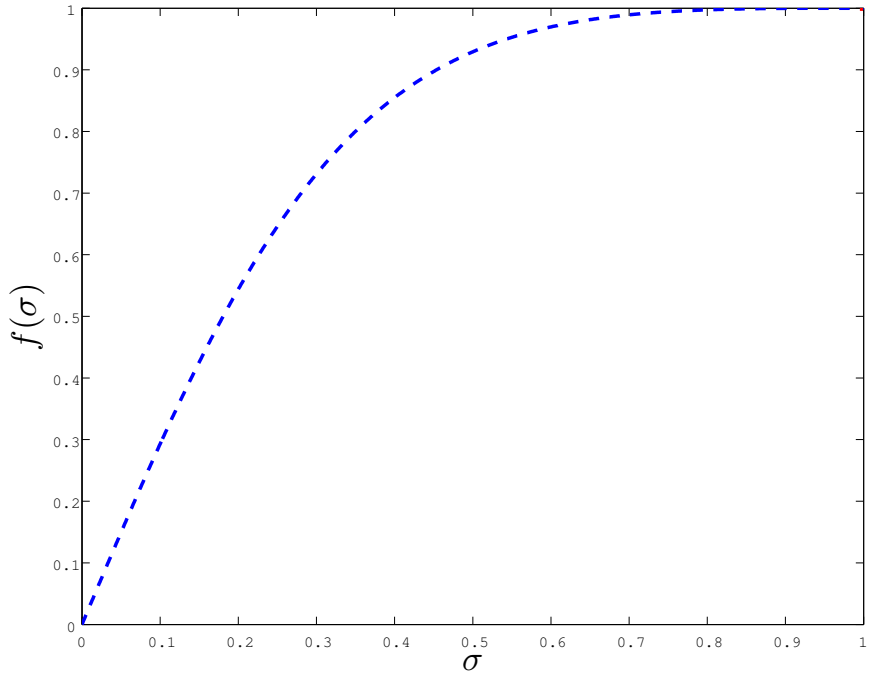
IRS: static parameters

it=3



QDWH: dynamical parameters

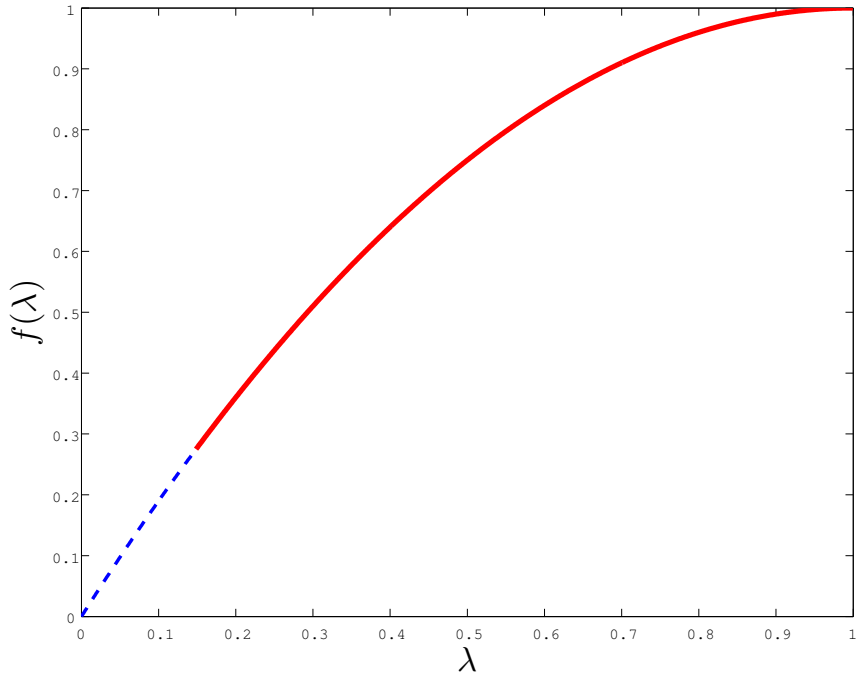
it=2



IRS vs. QDWH convergence

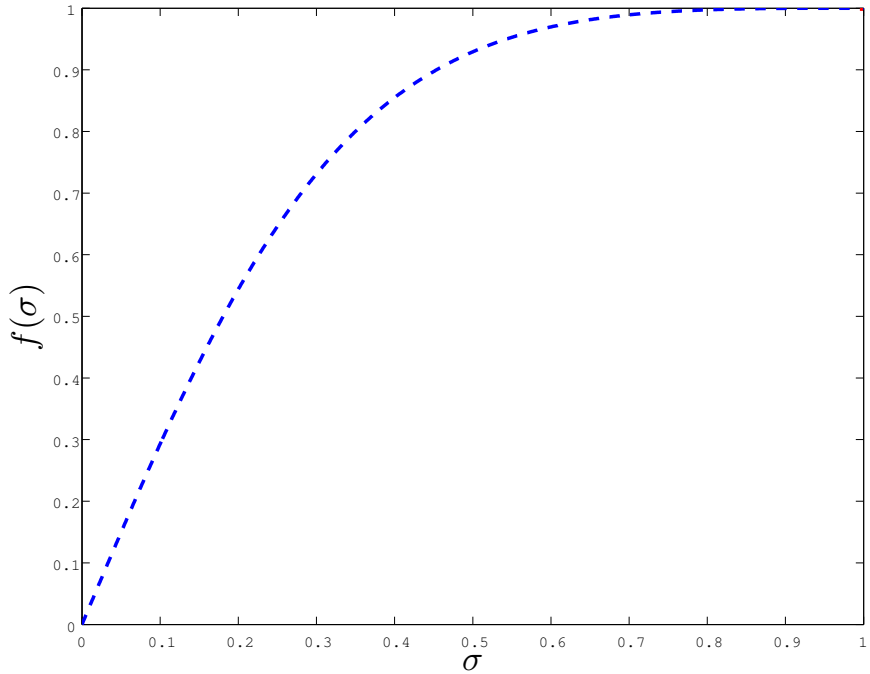
IRS: static parameters

it=4



QDWH: dynamical parameters

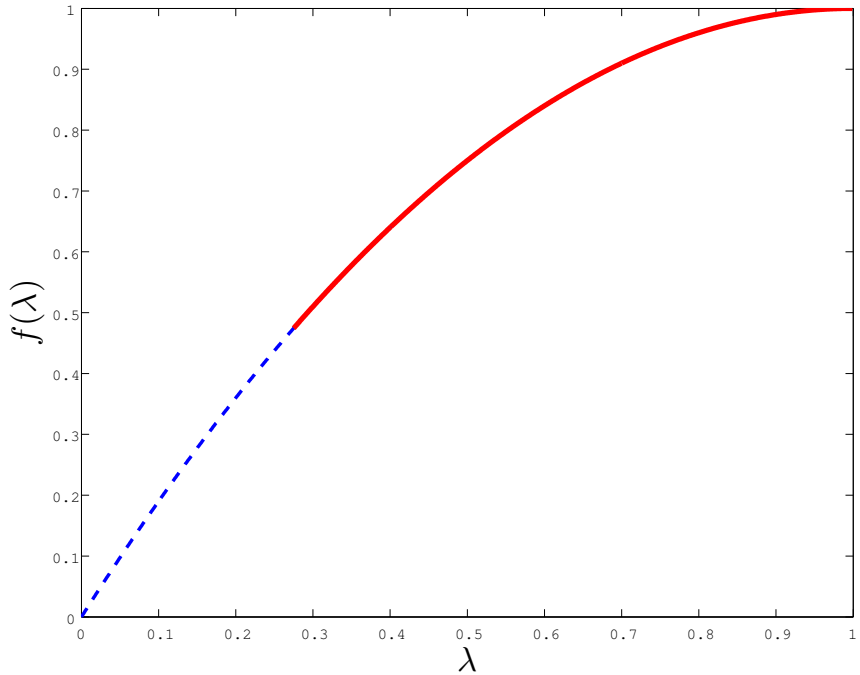
it=2



IRS vs. QDWH convergence

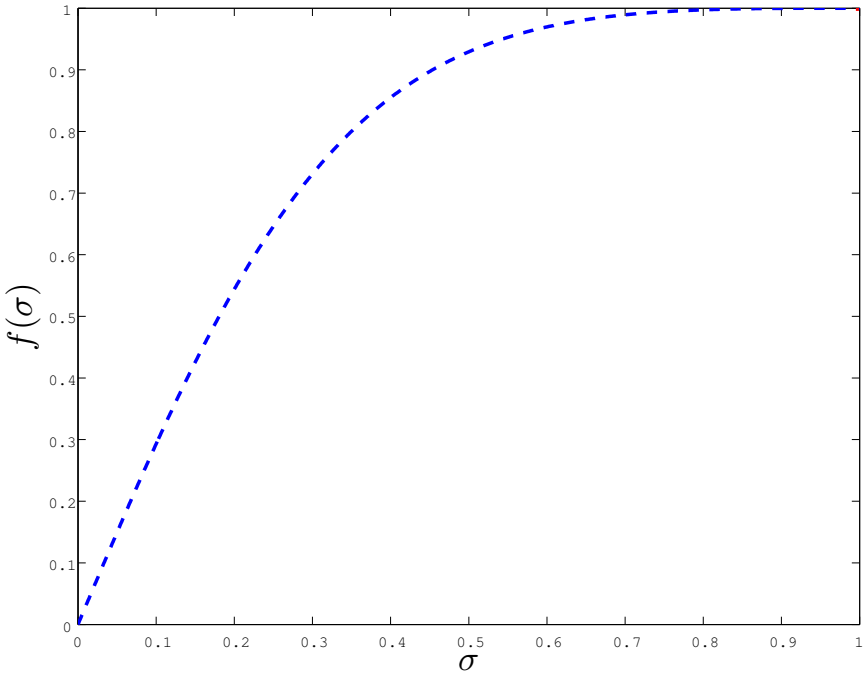
IRS: static parameters

it=5



QDWH: dynamical parameters

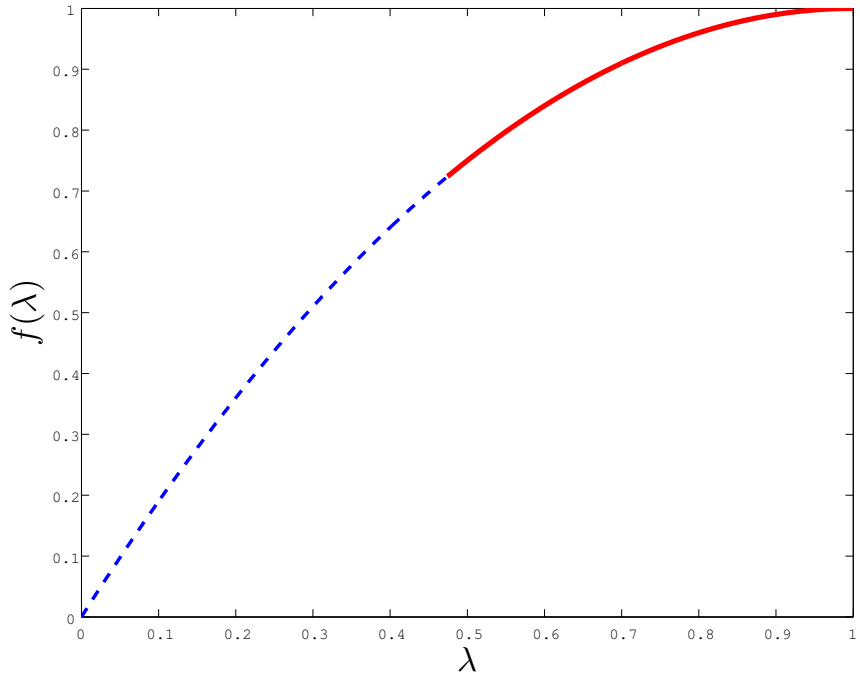
it=2



IRS vs. QDWH convergence

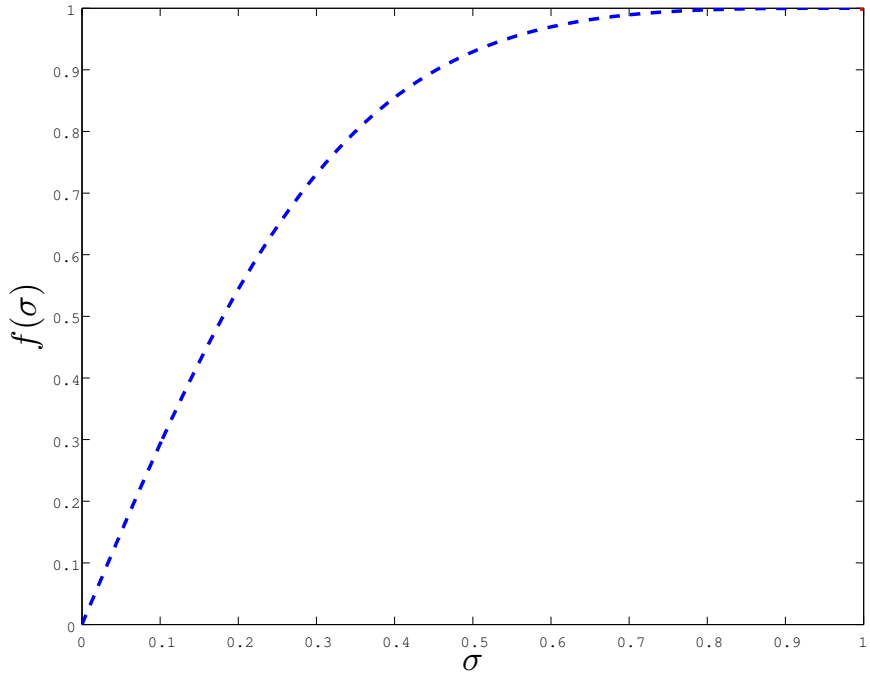
IRS: static parameters

it=6



QDWH: dynamical parameters

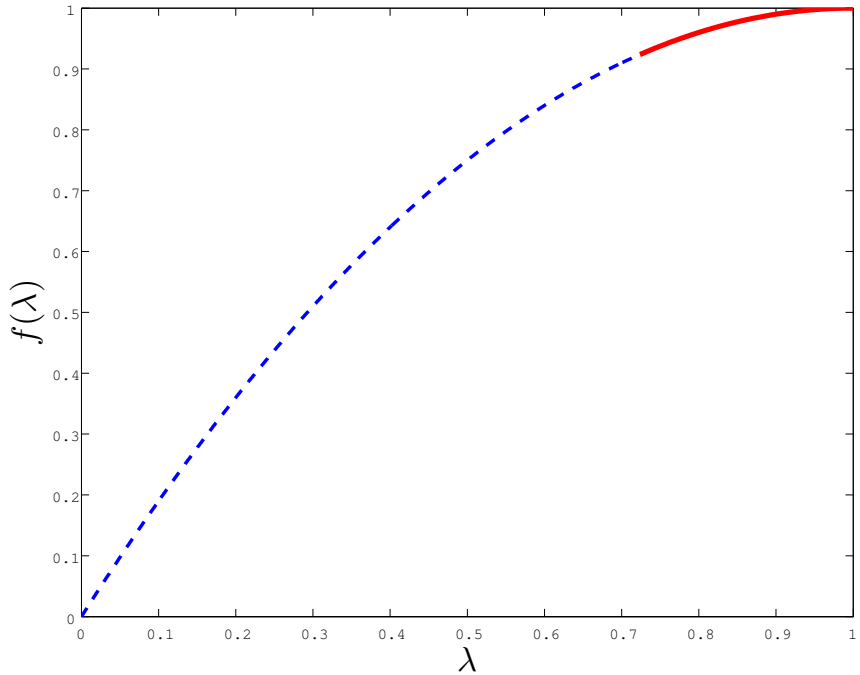
it=2



IRS vs. QDWH convergence

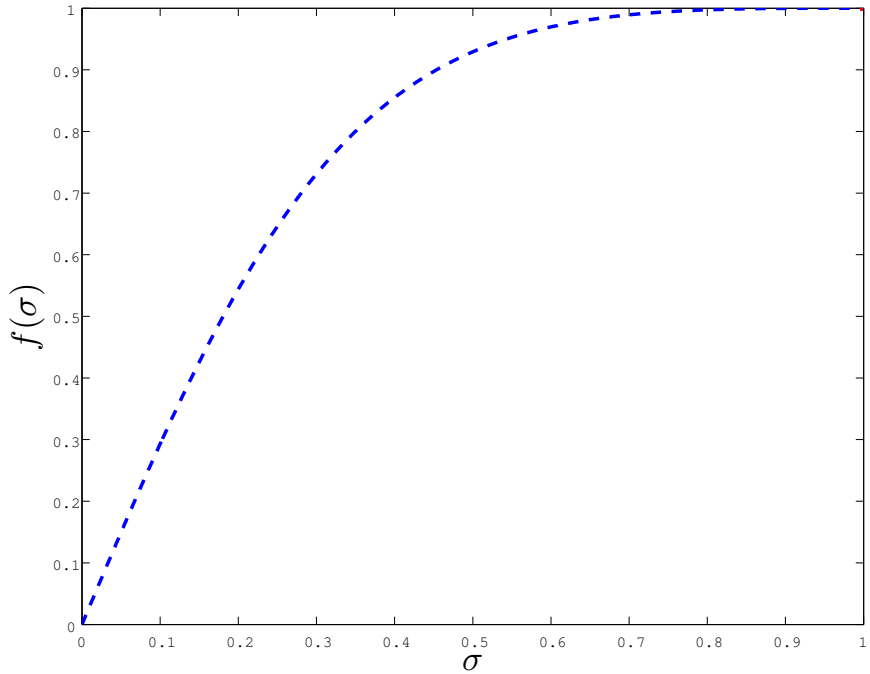
IRS: static parameters

it=7



QDWH: dynamical parameters

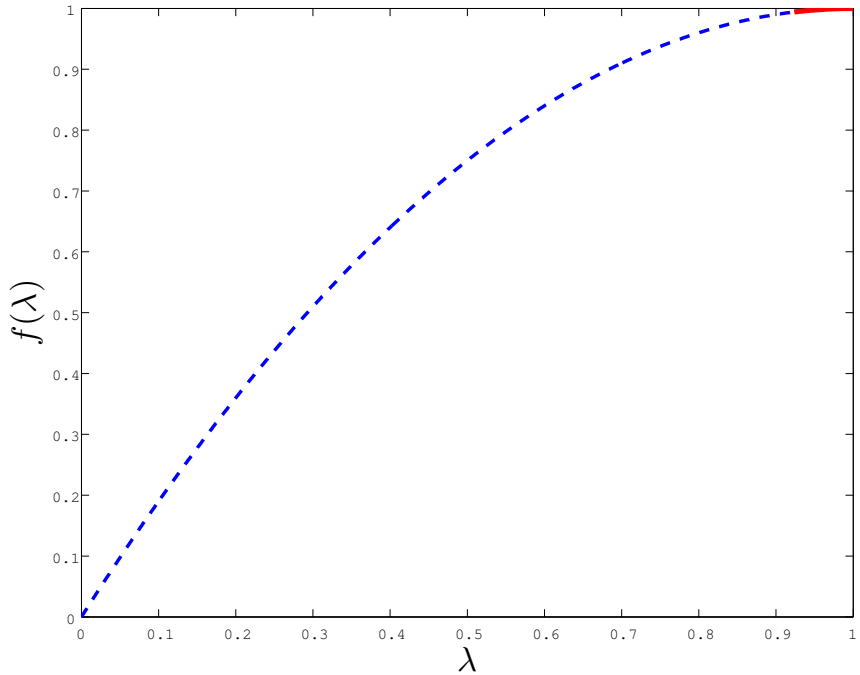
it=2



IRS vs. QDWH convergence

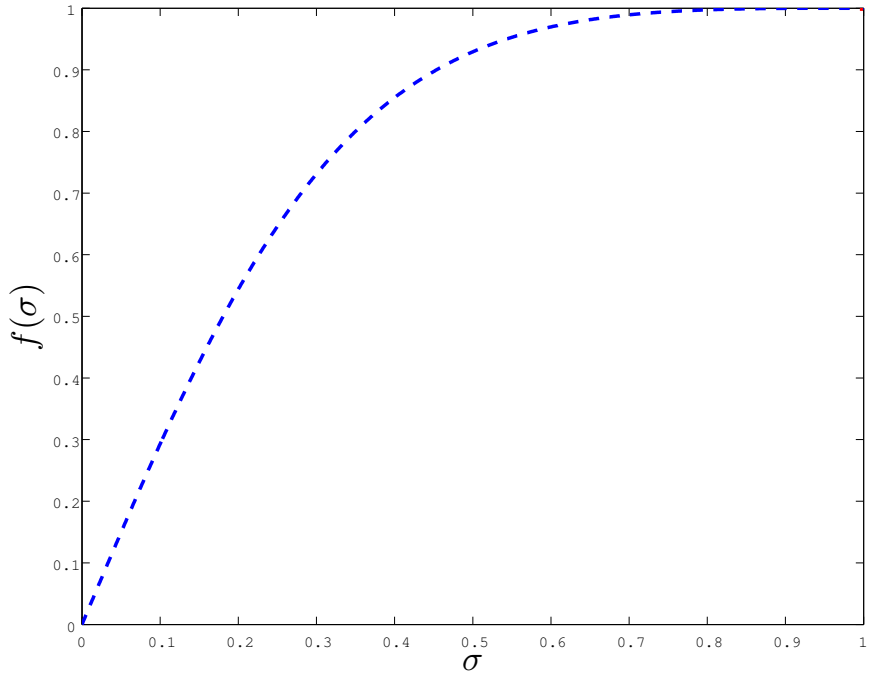
IRS: static parameters

it=8



QDWH: dynamical parameters

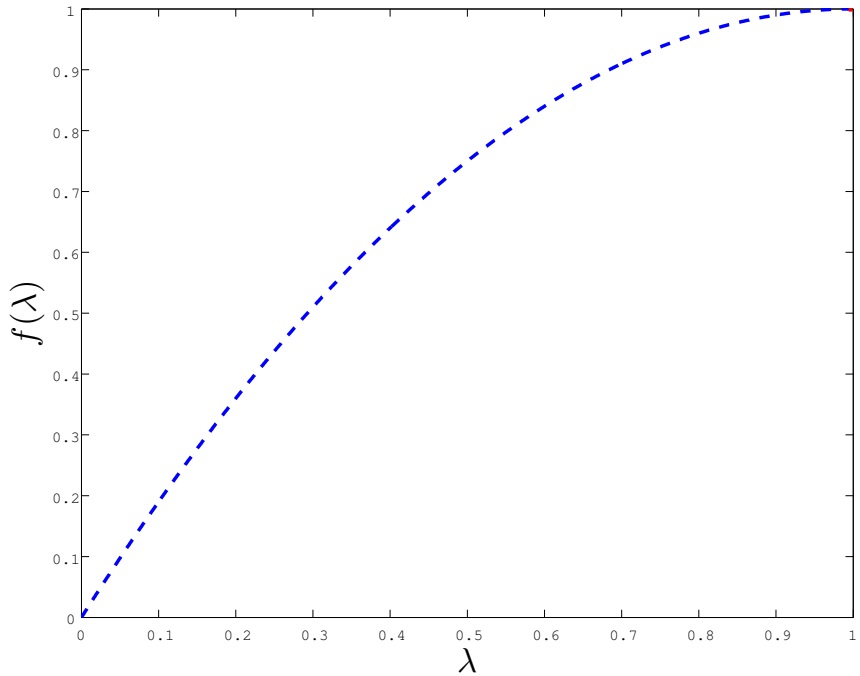
it=2



IRS vs. QDWH convergence

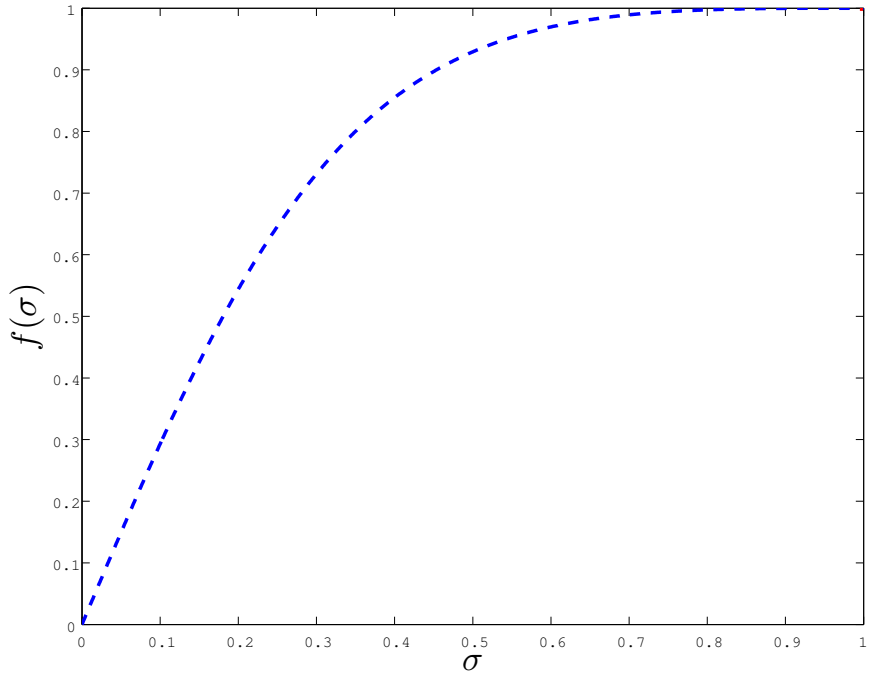
IRS: static parameters

it=9



QDWH: dynamical parameters

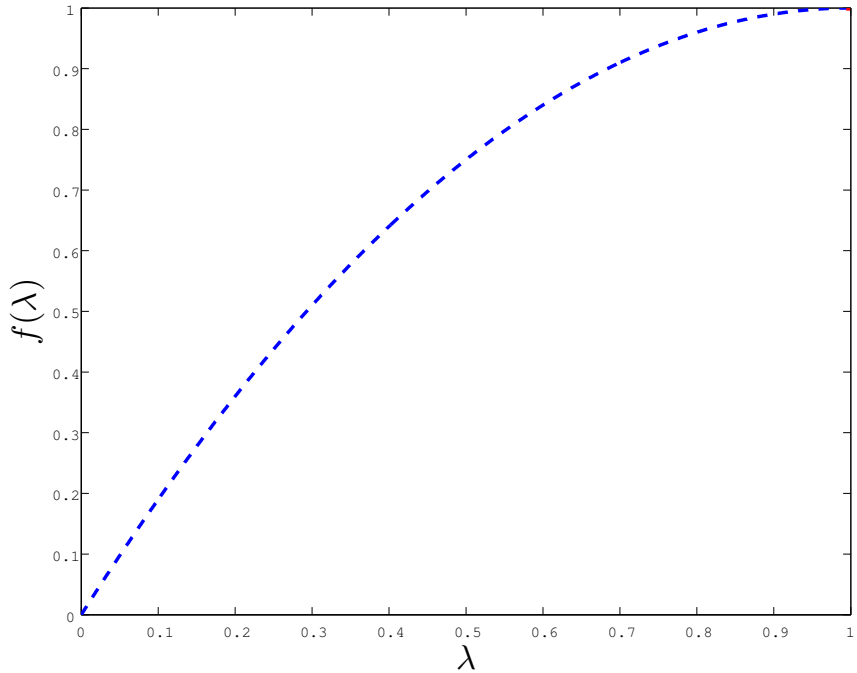
it=2



IRS vs. QDWH convergence

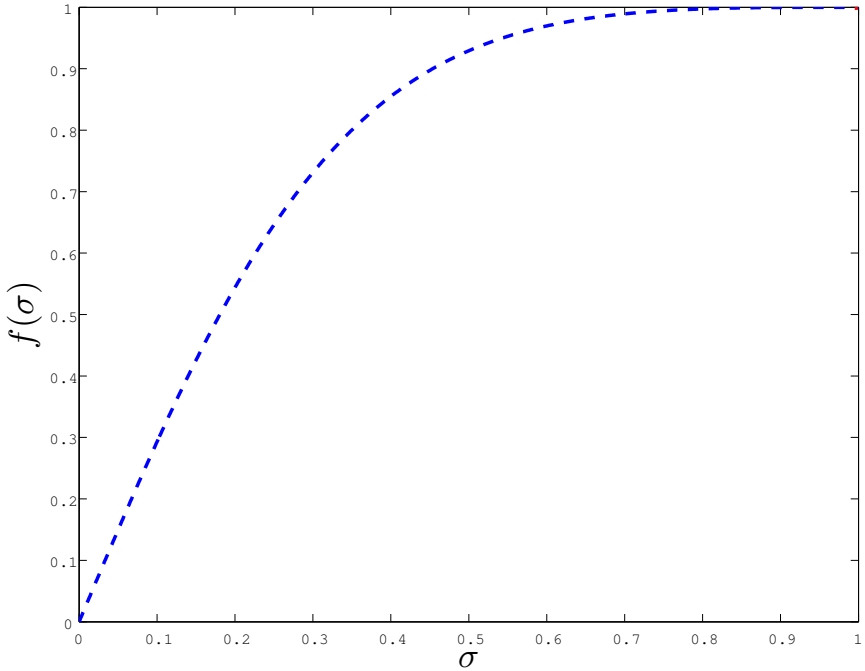
IRS: static parameters

it=9



QDWH: dynamical parameters

it=2



dynamical parameters speed up convergence significantly

In fact, no more than 6 iterations are needed for QDWH for any A with $\kappa_2(A) < 10^{16}$

Polar decomposition + symmetric eigenproblem = SVD

Algorithm **SVD**

1. compute polar decomposition

$$A = U_P H \tag{2}$$

2. compute symmetric eigendecomposition

$$H = V^T \Sigma V \tag{3}$$

3. get SVD $A = (U_P V^T)\Sigma V = U \Sigma V^T$

Polar decomposition + symmetric eigenproblem = SVD

Algorithm **SVD**

1. compute polar decomposition

$$A = U_P H \tag{2}$$

2. compute symmetric eigendecomposition

$$H = V^T \Sigma V \tag{3}$$

3. get SVD $A = (U_P V^T)\Sigma V = U \Sigma V^T$

- The computed SVD is backward stable if both (2) and (3) are

–[Higham and Papadimitriou (1993)]

$\Rightarrow$ **Backward stability again rests on that of polar decomposition**

- For $n \times n$ matrices, SVD is less expensive than **2** runs of QDWH-eig

note: SVD via eig $\left(\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}\right)$ costs $\simeq \times 8$

Comparison

- symmetric eigenproblem

	standard	ZZY	IRS	QDWH-eig
Arithmetic cost	$9n^3$	$\simeq 370n^3$	$\approx 720n^3$	$27n^3$
Backward stable?	$\surd$	unknown	$\times$	$\surd$
Minimize communication?	$\times$	$\surd$	$\surd$	$\surd$

- SVD

	standard	IRS	QDWH-SVD
Arithmetic cost	$26n^3$	$\simeq 5700n^3$	$34n^3 \sim 52n^3$
Backward stable?	$\surd$	$\times$	$\surd$
Minimize communication?	$\times$	$\surd$	$\surd$

nb: IRS is applicable to generalized non-symmetric eigenproblems

Experiments: spectral d-c algorithms for symeig

$$A = A^T \in \mathbb{R}^{100\times 100},$$

$\lambda(A) = \{1, r, r^2, \ldots, r^{n-1}\}$ with $r = -\kappa^{-1/(n-1)}$, 100 runs

$\kappa_2(A)$		10^2		10^8		10^{15}	
		min	max	min	max	min	max
iter	QDWH-eig	4	5	5	6	6	6
	ZZY	12	12	32	32	55	55
	IRS	12	12	32	32	54	55
back error	QDWH-eig	4.2e-16	5.1e-16	4.3e-16	5.5e-16	5.0e-16	6.3e-16
	ZZY	1.3e-15	1.6e-15	1.9e-15	2.5e-15	2.2e-15	3.1e-15
	IRS	1.9e-15	4.2e-13	3.6e-13	3.4e-11	5.4e-10	1.1e-6

Numerical experiments: symeig accuracy

NAG MATLAB toolbox experiments with n-by-n matrices $A^T = A$, uniformly distributed eigenvalues

Backward error $\|A - \widehat{V}\widehat{\Lambda}\widehat{V}^T\|_F / \|A\|_F$

	$n = 2000$	$n = 4000$	$n = 6000$	$n = 8000$	$n = 10000$
QDWH-eig	2.1e-15	2.4e-15	2.6e-15	2.8e-15	2.9e-15
Divide-Conquer	5.3e-15	7.3e-15	8.7e-15	9.8e-15	1.1e-14
MATLAB	1.4e-14	1.8e-14	2.3e-14	2.7e-14	3.0e-14
MR^3	4.2e-15	5.8e-15	7.1e-15	8.1e-15	8.9e-15
QR	1.5e-14	2.8e-14	2.5e-14	2.9e-14	3.2e-14

Distance from orthogonality $\|\widehat{V}^T\widehat{V} - I\|_F / \sqrt{n}$.

	$n = 2000$	$n = 4000$	$n = 6000$	$n = 8000$	$n = 10000$
QDWH-eig	7.7e-16	8.0e-16	8.2e-16	8.4e-16	8.5e-16
Divide-Conquer	4.6e-15	6.3e-15	7.6e-15	8.7e-15	9.6e-15
MATLAB	1.1e-14	1.5e-14	1.9e-14	2.1e-14	2.3e-14
MR^3	2.6e-15	3.6e-15	4.2e-15	4.8e-15	5.3e-15
QR	1.1e-14	2.4e-14	1.9e-14	2.2e-14	2.5e-14

Numerical experiments: symeig runtime

NAG MATLAB toolbox experiments with n-by-n matrices $A^T = A$, uniform eigenvalues

	Runtime(s)				
	$n = 2000$	$n = 4000$	$n = 6000$	$n = 8000$	$n = 10000$
QDWH-eig	11.0	64.1	188	526	959
Divide-Conquer	2.0	17.8	60.8	141	275
MATLAB	2.4	18.8	64.9	151	290
MR^3	4.0	37.0	120.7	262	504
QR	11.8	105.4	354.7	834	1631

Numerical experiments: SVD accuracy

NAG MATLAB toolbox experiments with n-by-n MATLAB randsvd matrices with uniform singular values, $\kappa_2(A) = 1.5$

Backward error $\|A - \widehat{U}\widehat{\Sigma}\widehat{V}^T\|_F / \|A\|_F$

	$n = 2000$	$n = 4000$	$n = 6000$	$n = 8000$	$n = 10000$
QDWH-SVD	2.0e-15	2.3e-15	2.5e-15	2.7e-15	2.8e-15
Divide-Conquer	5.6e-15	7.8e-15	9.2e-15	1.0e-14	1.2e-14
MATLAB	5.8e-15	7.8e-15	9.2e-15	1.1e-14	1.2e-14
QR	1.6e-14	2.2e-14	2.7e-14	3.1e-14	3.5e-14

Distance from orthogonality $\max\{\|\widehat{U}^T\widehat{U} - I\|_F / \sqrt{n}, \|\widehat{V}^T\widehat{V} - I\|_F / \sqrt{n}\}$.

	$n = 2000$	$n = 4000$	$n = 6000$	$n = 8000$	$n = 10000$
QDWH-SVD	7.7e-16	8.0e-16	8.2e-16	8.4e-16	8.5e-16
Divide-Conquer	5.2e-15	7.2e-15	8.4e-15	9.5e-15	1.1e-14
MATLAB	8.1e-15	7.1e-15	8.4e-15	9.8e-15	1.1e-14
QR	1.2e-14	1.7e-14	2.1e-14	2.5e-14	2.8e-14

Numerical experiments: SVD runtime

NAG MATLAB toolbox experiments with n-by-n MATLAB randsvd matrices with uniform singular values, $\kappa_2(A) = 1.5$

	Runtime(s)				
	$n = 2000$	$n = 4000$	$n = 6000$	$n = 8000$	$n = 10000$
QDWH-SVD	15.4	90.3	269	581	1079
Divide-Conquer	5.9	43.5	140	316	605
MATLAB	4.0	32.0	105	243	470
QR	20.5	151.8	458	1071	2010

Summary and future work

- Proposed spectral divide-and-conquer algorithms for symeig, SVD
  - Arithmetic cost within a factor 3 of standard algorithms.
  - Backward stability is proved and empirically better than standard algorithms.
  - Uses only matrix multiplication, QR and Cholesky, hence minimizes communication.

for details, see –[Nakatsukasa and Higham, MIMS EPrint May 2012]
(codes also available at MATLAB Central Exchange 36830)

Summary and future work

- Proposed spectral divide-and-conquer algorithms for symeig, SVD
  - Arithmetic cost within a factor 3 of standard algorithms.
  - Backward stability is proved and empirically better than standard algorithms.
  - Uses only matrix multiplication, QR and Cholesky, hence minimizes communication.

for details, see –[Nakatsukasa and Higham, MIMS EPrint May 2012]
(codes also available at MATLAB Central Exchange 36830)

Work in progress

- Higher-order iteration using Zolotarev's rational approximation
  $\rightarrow$ further enhanced parallelizability

–[with R. W. Freund]

- Performance benchmarking with a parallel implementation

–[with J. Poulson, G. Quintana, R. van de Geijn]

- Extension to generalized eigenvalue problems

–[with N. J. Higham, F. Tisseur]

Spectral d-c for generalized eigenproblem

Goal: given (A, B), compute nonsingular Q, Z such that

$$(QAZ, QBZ) = \left(\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}, \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \right)$$

$$= \left(\begin{bmatrix} A_{11} & \\ & A_{22} \end{bmatrix}, \begin{bmatrix} B_{11} & \\ & B_{22} \end{bmatrix} \right).$$

Spectral d-c for generalized eigenproblem: outline

Goal: given (A, B), compute nonsingular Q, Z such that

$$QAZ = \begin{bmatrix} A_{11} & \\ & A_{22} \end{bmatrix}, \quad QBZ = \begin{bmatrix} B_{11} & \\ & B_{22} \end{bmatrix}. \tag{4}$$

Idea: map eigenvalues to ± 1; $Re(\lambda) > 0$ to 1, $Re(\lambda) < 0$ to -1.

Suppose (A, B) diagonalizable: $A = X \begin{bmatrix} \Lambda_+ & \\ & \Lambda_- \end{bmatrix} Y, \quad B = XY.$

1. Compute (NB: left eigenmatrix is $W \neq X$)

$$\widetilde{A} = W \begin{bmatrix} I & \\ & -I \end{bmatrix} Y, \qquad \widetilde{B} = WY.$$

2. Form

$$(\widetilde{A} + \widetilde{B})^* = Y^* \begin{bmatrix} 2I & \\ & 0 \end{bmatrix} W^* = [Q_{1+}\ Q_{2+}] \begin{bmatrix} R_+ \\ 0 \end{bmatrix},$$

$$-(\widetilde{A} - \widetilde{B})^* = Y^* \begin{bmatrix} 0 & \\ & 2I \end{bmatrix} W^* = [Q_{1-}\ Q_{2-}] \begin{bmatrix} R_- \\ 0 \end{bmatrix}.$$

3. Then letting $Z = [Q_{2+}, Q_{2-}]$, $\exists$ nonsingular $Q = [Q_a, Q_b]$ such that (4) holds (Q_a, Q_b computed via QR)

Mapping eigenvalues to ± 1

Given

$$A = X \begin{bmatrix} \Lambda_+ & \\ & \Lambda_- \end{bmatrix} Y, \quad B = XY,$$

the goal is to compute

$$\widetilde{A} = W \begin{bmatrix} I & \\ & -I \end{bmatrix} Y, \quad \widetilde{B} = WY.$$

We do this via computing

$$A_k = W_k \begin{bmatrix} f_k(\Lambda_{+,k-1}) & \\ & f_k(\Lambda_{-,k-1}) \end{bmatrix} Y, \quad B_k = W_k Y,$$

where $f_k(\cdots f_2(f_1(x)) \to \text{sign}(x)\ (= \pm 1)$ for $x \in \Lambda(A, B)$.

- $f(x)$ are the same functions as in QDWH, has the form

$$\begin{aligned} f(x) = \frac{x(a + bx^2)}{1 + cx^2} &= \frac{b}{c}x + \left(a - \frac{b}{c}\right)\frac{x}{1 + cx^2} \\ &= \frac{b}{c}x + \frac{1}{2i}\left(a - \frac{b}{c}\right)\left(\frac{x}{\sqrt{c}x + i} - \frac{x}{\sqrt{c}x - i}\right) \end{aligned}$$

Mapping eigenvalues of (A, B) by rational functions

Let

$$\begin{bmatrix} \alpha A + \beta B \\ -(\gamma A + \delta B) \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}.$$

Then

$$\begin{bmatrix} Q_{12}^T & Q_{22}^T \end{bmatrix} \begin{bmatrix} \alpha A + \beta B \\ -(\gamma A + \delta B) \end{bmatrix} = Q_{12}^T(\alpha A + \beta B) - Q_{22}^T(\gamma A + \delta B) = 0,$$

that is,

$$(\alpha A + \beta B)(\gamma A + \delta B)^{-1} = Q_{12}^{-T} Q_{22}^T.$$

This means

$$A^{(1)} = Q_{22}^T A, \quad B^{(1)} = Q_{12}^T B,$$

satisfies (recall $A = X\Lambda Y, B = XY$)

$$A^{(1)} = W^{(1)} g(\Lambda) Y, \quad B^{(1)} = W^{(1)} Y, \quad \text{where} \quad g(x) = x\frac{\alpha x + \beta}{\gamma x + \delta}.$$

Forming $W_2(\Lambda_1+\Lambda_2, I)Y$ from $W_1(\Lambda_1, I)Y, W_2(\Lambda_2, I)Y$

Goal: obtain $\widetilde{A} = Wf(\Lambda)Y,\ \widetilde{B} = WY$, where $A = X\Lambda Y,\ B = XY$,

$$f(x) = \frac{b}{c}x + \frac{1}{2i}\left(a - \frac{b}{c}\right)\left(\frac{x}{\sqrt{c}x+i} - \frac{x}{\sqrt{c}x-i}\right) := \frac{b}{c}x + g_1(x) + g_2(x).$$

We have $A^{(i)} = W^{(i)}g_i(\Lambda)Y,\ B^{(i)} = W^{(i)}Y = W^{(i,B)}B,\ i = 1, 2$. Hence if

$$\begin{bmatrix} B^{(1)} \\ -B^{(2)} \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix},$$

then $Q_{22}^* W^{(2)} = Q_{12}^* W^{(1)} := W$, so $Q_{22}^* B^{(2)} = Q_{12}^* B^{(1)} = WY = \widetilde{B}$ and

$$\begin{aligned} \widetilde{A} :&= \frac{b}{c}Q_{12}^* W^{(1,B)}A + Q_{12}^* A^{(1)} + Q_{22}^* A^{(2)} \\ &= W(\frac{b}{c}\Lambda + g_1(\Lambda) + g_2(\Lambda))Y = Wf(\Lambda)Y \end{aligned}$$

give the required $(\widetilde{A}, \widetilde{B})$.

Properties: Spectral d-c for GEP

– When (A, B) are symmetric but indefinite, Spectral d-c takes full advantage of symmetry by congruence:

$$Z^* A Z = \begin{bmatrix} A_{11} & \\ & A_{22} \end{bmatrix}, \quad Z^* B Z = \begin{bmatrix} B_{11} & \\ & B_{22} \end{bmatrix}.$$

- Conventional approach uses the QZ algorithm:
  - Destroys symmetry
  - Yields upper-triangular (QAZ, QAB), not (block) diagonal
- No need to check for definiteness of (A, B)

– Spectral d-c uses (heavy use of)QR and matmul, minimizes communication

– Seems to work even for singular (A, B)

MATLAB experiments with 1000×1000 matrices

Showing backward error $\max\left(\frac{\|\widehat{P}A\widehat{Q}-\widehat{\Lambda}\|_F}{\|\widehat{\Lambda}\|_F}, \frac{\|\widehat{P}B\widehat{Q}-I\|_F}{\sqrt{n}}\right)$.

- A, B symmetric, $B > 0$, $\kappa_2(B) = 10^{10}$.

	backward error
QDWH-GEP	2.7e-13
Matlab eig	6.9e-11

- A, B symmetric indefinite, $B = X^T \mathrm{diag}(\pm 1)X$, $\kappa_2(B) = 10^{10}$.

	backward error
QDWH-GEP	2.7e-13
Matlab eig	5.0e-11

- A, B nonsymmetric, 998 real eigenvalues

	backward error
QDWH-GEP	3.4e-13
Matlab eig	1.8e-10

References

- G. Ballard, J. Demmel, I. Dumitriu, Minimizing communication for eigenproblems and the singular value decomposition. LAWN237.
- Y. Nakatsukasa, Z. Bai, and F. Gygi. Optimizing Halley's iteration for computing the matrix polar decomposition. SIAM J. Matrix Anal. Appl., 2010.
- Y. Nakatsukasa and N. J. Higham. Backward stability of iterations for computing the polar decomposition. SIAM J. Matrix Anal. Appl., 2012.
- Y. Nakatsukasa and N. J. Higham. Efficient, communication-minimizing algorithms for the symmetric eigenvalue decomposition and the singular value decomposition. MIMS EPrint 2012.52.

Standard algorithm for SVD

1. Reduce matrix to bidiagonal form

$$A = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \overset{H_L}{\rightarrow} \begin{bmatrix} * & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix} \overset{H_R}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix}$$

$$\overset{H_L}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & * \\ & & * & * \\ & & * & * \end{bmatrix} \overset{H_R}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & \\ & & * & * \\ & & * & * \end{bmatrix} \overset{H_L}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & \\ & & * & * \\ & & & * \end{bmatrix} \equiv B$$

–[Golub and Kahan (1965)]

Accumulate orthogonal factors: $A = U_A B V_A^H$ where $U_A = (\prod H_L)^H, V_A = \prod H_R$

2. Compute SVD $B = U_B \Sigma V_B^H$ (divide-conquer, QR, dqds + twisted factorization)
3. SVD: $A = (U_A U_B)\Sigma(V_B^H V_A^H) = U \Sigma V^H$

Standard algorithm for SVD

1. Reduce matrix to bidiagonal form
   $\Rightarrow$ Expensive in communication cost

$$A = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \overset{H_L}{\rightarrow} \begin{bmatrix} * & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix} \overset{H_R}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix}$$

$$\overset{H_L}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & * \\ & & * & * \\ & & * & * \end{bmatrix} \overset{H_R}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & \\ & & * & * \\ & & * & * \end{bmatrix} \overset{H_L}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & \\ & & * & * \\ & & & * \end{bmatrix} \equiv B$$

–[Golub and Kahan (1965)]

   Accumulate orthogonal factors: $A = U_A B V_A^H$ where $U_A = (\prod H_L)^H, V_A = \prod H_R$
2. Compute SVD $B = U_B \Sigma V_B^H$ (divide-conquer, QR, dqds + twisted factorization)
3. SVD: $A = (U_A U_B)\Sigma(V_B^H V_A^H) = U \Sigma V^H$

Standard algorithm for SVD

1. Reduce matrix to bidiagonal form
   $\Rightarrow$ Expensive in communication cost

$$A = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \overset{H_L}{\rightarrow} \begin{bmatrix} * & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix} \overset{H_R}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix}$$

$$\overset{H_L}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & * \\ & & * & * \\ & & * & * \end{bmatrix} \overset{H_R}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & \\ & & * & * \\ & & * & * \end{bmatrix} \overset{H_L}{\rightarrow} \begin{bmatrix} * & * & & \\ & * & * & \\ & & * & * \\ & & & * \end{bmatrix} \equiv B$$

–[Golub and Kahan (1965)]

   Accumulate orthogonal factors: $A = U_A B V_A^H$ where $U_A = (\prod H_L)^H, V_A = \prod H_R$
2. Compute SVD $B = U_B \Sigma V_B^H$ (divide-conquer, QR, dqds + twisted factorization)
3. SVD: $A = (U_A U_B)\Sigma(V_B^H V_A^H) = U\Sigma V^H$

We develop completely different algorithms based on

spectral divide-and-conquer

Backward stability proof of QDWH-eig

Goal: show $\|E\|_2 = \|\epsilon A\|_2$ where $\widehat{V}^T A \widehat{V} = \begin{bmatrix} A_+ & E^T \\ E & A_- \end{bmatrix}$

Assumptions:

$$A = \widehat{U}\widehat{H} + \epsilon\|A\|_2, \quad \widehat{U}^T\widehat{U} - I = \epsilon.$$

$$\widehat{V}^T \begin{bmatrix} I & \\ & -I \end{bmatrix} \widehat{V} = \widehat{U} + \epsilon$$

- By the assumptions $A = \widehat{V}\begin{bmatrix} I & \\ & -I \end{bmatrix}\widehat{V}^T\widehat{H} + \epsilon\|A\|_2$, so

$$\begin{aligned} 0 &= A - A^T \\ &= \left(\widehat{V}\begin{bmatrix} I & \\ & -I \end{bmatrix}\widehat{V}^T\widehat{H} - \widehat{H}^T\widehat{V}\begin{bmatrix} I & \\ & -I \end{bmatrix}\widehat{V}^T\right) + \epsilon\|A\|_2 \end{aligned}$$

- Therefore

$$\epsilon\|A\|_2 = \widehat{V}^T\widehat{H}\widehat{V} - \begin{bmatrix} I & \\ & -I \end{bmatrix}\widehat{V}^T\widehat{H}\widehat{V}\begin{bmatrix} I & \\ & -I \end{bmatrix} = 2\begin{bmatrix} 0 & E^T \\ -E & 0 \end{bmatrix}$$

Subspace iteration to obtain V_+

$$U = [V_+ \; V_-] \begin{bmatrix} I & \\ & -I \end{bmatrix} [V_+ \; V_-]^T$$

- U is Hermitian with eigenvalues ± 1
- $\frac{1}{2}(U + I) = [V_+ \; V_-] \begin{bmatrix} I & \\ & 0 \end{bmatrix} [V_+ \; V_-]^T = V_+ V_+^T$ is the projection onto the positive eigenspace of A

Subspace iteration to obtain V_+

$$U = [V_+ \; V_-] \begin{bmatrix} I & \\ & -I \end{bmatrix} [V_+ \; V_-]^T$$

- U is Hermitian with eigenvalues ± 1
- $\frac{1}{2}(U + I) = [V_+ \; V_-] \begin{bmatrix} I & \\ & 0 \end{bmatrix} [V_+ \; V_-]^T = V_+ V_+^T$ is the projection onto the positive eigenspace of A

Obtain V_+ from $C \equiv \frac{1}{2}(U + I) = V_+ V_+^T$:

Subspace iteration to obtain V_+

$$U = [V_+ \; V_-] \begin{bmatrix} I & \\ & -I \end{bmatrix} [V_+ \; V_-]^T$$

- U is Hermitian with eigenvalues ± 1
- $\frac{1}{2}(U + I) = [V_+ \; V_-] \begin{bmatrix} I & \\ & 0 \end{bmatrix} [V_+ \; V_-]^T = V_+ V_+^T$ is the projection onto the positive eigenspace of A

Obtain V_+ from $C \equiv \frac{1}{2}(U + I) = V_+ V_+^T$:

- Simultaneous iteration with initial guess $Q_1 = C(:, 1 : \sqrt{\|Q\|_F^2 + 1})$
  $C = Q_1 R$, $CQ_1 = Q_2 R$, ...
  - Theoretical convergence factor is $\simeq 1/\epsilon$
  - Stop when $\widehat{C\widehat{V}_1} = \widehat{V}_1 + \epsilon$ and $\widehat{C\widehat{V}_2} = \epsilon$: in practice, at most 2 iterations are necessary
  - Use a random initial guess if fails (never happened in experiments)

Newton–Schulz postprocessing to improve orthogonality

In practice, the computed $\widehat{U}, \widehat{V}$ in $A \simeq \widehat{U}\Sigma\widehat{V}^*$ have distance from orthogonality $\|\widehat{U}^*\widehat{U} - I\|_F, \|\widehat{V}^*\widehat{V} - I\|_F > 0$.

A (empirically) powerful way to improve the orthogonality is to run a step of the Newton–Schulz iteration $U := \frac{1}{2}U(3I - U^*U)$.

$\max\{\|\widehat{U}^*\widehat{U} - I\|_F/\sqrt{n}, \|\widehat{V}^*\widehat{V} - I\|_F/\sqrt{n}\}$

	$n = 2000$	$n = 4000$	$n = 6000$	$n = 8000$	$n = 10000$
QR	1.2e-14	1.7e-14	2.1e-14	2.5e-14	2.8e-14
QDWH before N–S	2.8e-15	3.1e-15	3.4e-15	3.6e-15	3.8e-15
QDWH after N–S	7.7e-16	8.0e-16	8.2e-16	8.4e-16	8.5e-16

Faster iterations using Cholesky factorization

$$X_{k+1} = X_k(a_k I + b_k X_k^H X_k)(I + c_k X_k^H X_k)^{-1}, \quad X_0 = A/\alpha \tag{5}$$

- The QR-kernel

$$\begin{bmatrix} \sqrt{c_k} X_k \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \quad X_{k+1} = \frac{b_k}{c_k} X_k + \frac{1}{\sqrt{c_k}} \left(a_k - \frac{b_k}{c_k} \right) Q_1 Q_2^H$$

Note $\kappa_2 \left(\begin{bmatrix} \sqrt{c_k} X_k \\ I \end{bmatrix} \right) \leq \sqrt{c_k} + 1$, since $\|X_k\|_2 \leq 1$

- well-conditioned if c_k is not too large (e.g., $c_k \leq 100$), then **always** true for $k \geq 3$
- then, safely use CholeskyQR to compute Q_1, Q_2:

$$Z = I + c_k X_k^H X_k, \quad W = chol(Z),$$
$$Q_1 = \sqrt{c_k} X_k W^{-1}, \quad Q_2 = W^{-1}$$

communication-optimal + fewer arithmetic

Backward stability of iterations for polar decomposition

–[Nakatsukasa and Higham, SIMAX 2012]

An iteration $X_{k+1} = f_k(X_k), \quad X_0 = A, \lim_{k\to\infty} X_k = U$ for computing the polar decomposition is backward stable ($A - \widehat{U}\widehat{H} = \epsilon\|A\|$) if

1. Each iteration is mixed backward–forward stable:
   $\widehat{X}_{k+1} = f_k(\widetilde{X}_k) + \epsilon\|\widehat{X}_{k+1}\|, \quad \widetilde{X}_k = X_k + \epsilon\|X_k\|$

2. The mapping function $f_k(x)$ does not significantly decrease the relative size of σ_i for $i = 1 : n$

$$d\frac{f_k(\sigma_i)}{\|f_k(\widetilde{X})\|_2} \geq \frac{\sigma_i}{\|\widetilde{X}\|_2}$$

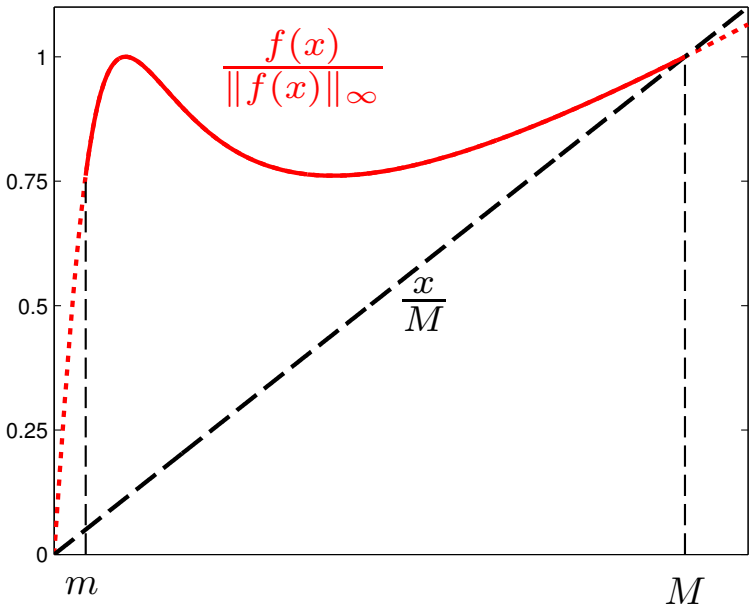
for $d = O(1)$.

(we prove $A - \widehat{U}\widehat{H} = d\epsilon\|A\|$, $H - \widehat{H} = d\epsilon\|H\|$)

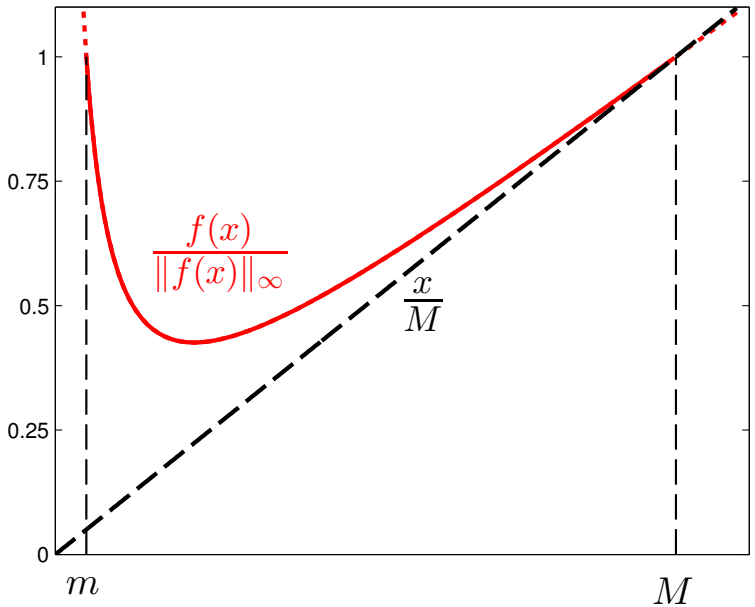
– for details, please go to Nick Higham's talk (MS 40, Wed 12:15)

Mapping function: stable (top) and unstable (bottom)

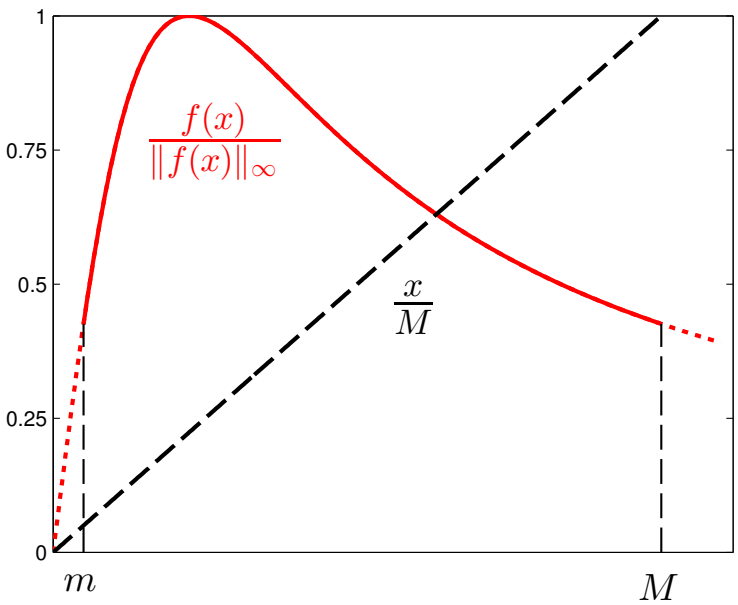
QDWH: $d = 1$



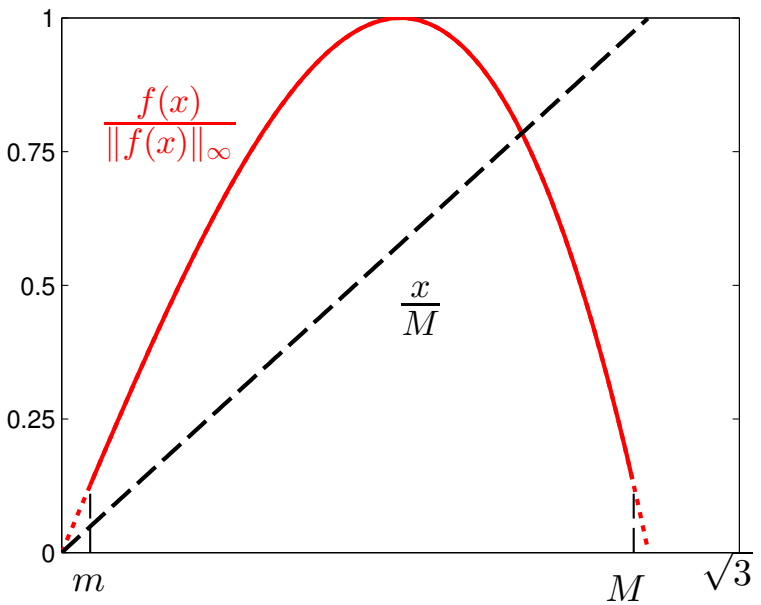
scaled Newton: $d = 1$



inverse Newton: $d \geq (\kappa_2(A))^{1/2}$



Newton–Schulz



Scaled Newton iteration

$$X_{k+1} = \frac{1}{2}(\zeta_k X_k + (\zeta_k X_k)^{-*}), \quad X_0 = A$$

– $\lim_{k\to\infty} X_k = U$ of $A = UH$ quadratically with appropriate scalings ζ_k

– Higham(86) : $\zeta_k = \left(\frac{\|X_k^{-1}\|_1 \|X_k^{-1}\|_\infty}{\|X_k\|_1 \|X_k\|_\infty} \right)^{1/4}$

Byers-Xu (08) : $\zeta_{k+1} = \frac{2}{\sqrt{(\zeta_k + 1/\zeta_k)}}$, $\zeta_0 = 1/\sqrt{ab}$, $a \geq \|A\|_2, b \leq \sigma_{\min}(A)$

– Both scalings work well in practice

Scaled Newton iteration

$$X_{k+1} = \frac{1}{2}(\zeta_k X_k + (\zeta_k X_k)^{-*}), \quad X_0 = A$$

– $\lim_{k\to\infty} X_k = U$ of $A = UH$ quadratically with appropriate scalings ζ_k

– Higham(86) : $\zeta_k = \left(\frac{\|X_k^{-1}\|_1 \|X_k^{-1}\|_\infty}{\|X_k\|_1 \|X_k\|_\infty} \right)^{1/4}$

Byers-Xu (08) : $\zeta_{k+1} = \frac{2}{\sqrt{(\zeta_k + 1/\zeta_k)}}$, $\zeta_0 = 1/\sqrt{ab}$, $a \geq \|A\|_2, b \leq \sigma_{\min}(A)$

– Both scalings work well in practice

- **However, X_k^{-1} needed explicitly**
  - Efforts in designing inverse-free algorithms
    –[Bai, Demmel and Gu (97), Crudge (98), Byers and Xu (01)]
    1. Instability of computing inverses
    2. High communication cost (pivoting is expensive)

Scaled Newton iteration

$$X_{k+1} = \frac{1}{2}(\zeta_k X_k + (\zeta_k X_k)^{-*}), \quad X_0 = A$$

– $\lim_{k\to\infty} X_k = U$ of $A = UH$ quadratically with appropriate scalings ζ_k

– Higham(86) : $\zeta_k = \left(\frac{\|X_k^{-1}\|_1 \|X_k^{-1}\|_\infty}{\|X_k\|_1 \|X_k\|_\infty} \right)^{1/4}$

Byers-Xu (08) : $\zeta_{k+1} = \frac{2}{\sqrt{(\zeta_k + 1/\zeta_k)}}$, $\zeta_0 = 1/\sqrt{ab}$, $a \geq \|A\|_2, b \leq \sigma_{\min}(A)$

– Both scalings work well in practice

- **However, X_k^{-1} needed explicitly**
  - Efforts in designing inverse-free algorithms
    –[Bai, Demmel and Gu (97), Crudge (98), Byers and Xu (01)]
    1. Instability of computing inverses
    2. High communication cost (pivoting is expensive)

Known inverse-free algorithms faced instability or slow convergence

Halley iterations

Halley iteration:

$$X_{k+1} = X_k(3I + X_k^* X_k)(I + 3X_k^* X_k)^{-1}, \quad X_0 = A.$$

- $X_k \to U$ as $k \to \infty$, convergence is global and asymptotically cubic
- Can be implemented inverse-free via QR
- Applicable to rectangular/singular matrices
- Slow convergence for ill-conditioned A

Halley iterations

Halley iteration:

$$X_{k+1} = X_k(3I + X_k^* X_k)(I + 3X_k^* X_k)^{-1}, \quad X_0 = A.$$

- $X_k \to U$ as $k \to \infty$, convergence is global and asymptotically cubic
- Can be implemented inverse-free via QR
- Applicable to rectangular/singular matrices
- Slow convergence for ill-conditioned A

(New): Dynamically Weighted Halley iteration (DWH)

$$X_{k+1} = X_k(a_k I + b_k X_k^* X_k)(I + c_k X_k^* X_k)^{-1}, \quad X_0 = A/\alpha, \quad \alpha \simeq \|A\|_2$$

- Choose a_k, b_k, c_k dynamically to get "suboptimal" convergence speed

Choosing weighting parameters

$$X_{k+1} = X_k(a_k I + b_k X_k^* X_k)(I + c_k X_k^* X_k)^{-1}$$

Suppose $[\sigma_{\min}(X_k), \sigma_{\max}(X_k)] \subseteq [\ell_k, 1]$, then

- $\sigma_i(X_{k+1}) = f_k(\sigma_i(X_k))$, where $f_k(x) = x\dfrac{a_k + b_k x^2}{1 + c_k x^2}$
- $\sigma_i(X_{k+1}) \in [\min_{\ell_k \le x \le 1} f_k(x), \max_{\ell_k \le x \le 1} f_k(x)]$

Choosing weighting parameters

$$X_{k+1} = X_k(a_k I + b_k X_k^* X_k)(I + c_k X_k^* X_k)^{-1}$$

Suppose $[\sigma_{\min}(X_k), \sigma_{\max}(X_k)] \subseteq [\ell_k, 1]$, then

- $\sigma_i(X_{k+1}) = f_k(\sigma_i(X_k))$, where $f_k(x) = x\dfrac{a_k + b_k x^2}{1 + c_k x^2}$
- $\sigma_i(X_{k+1}) \in [\min_{\ell_k \le x \le 1} f_k(x), \max_{\ell_k \le x \le 1} f_k(x)]$

Idea: Find (a_k, b_k, c_k) such that

$$\max_{a_k, b_k, c_k > 0} \left\{ \min_{\ell_k \le x \le 1} f_k(x) \right\},$$

under $\quad 0 < f_k(x) \le 1 \quad$ for $\quad \ell_k \le x \le 1$

Solving rational optimization problem

$$\max_{a,b,c}\left\{\min_{\ell\le x\le 1} f(x)\right\} \quad \text{under} \quad f(x)=x\frac{a+bx^2}{1+cx^2}, \quad 0<f(x)\le 1 \quad \text{on} \quad [\ell,1]$$

- Analytic solution (a,b,c):

$$a=h(\ell), \quad b=(a-1)^2/4, \quad c=a+b-1,$$

where $h(\ell)=\sqrt{1+d}+\frac{1}{2}\sqrt{8-4d+\frac{8(2-\ell^2)}{\ell^2\sqrt{1+d}}}$, $d=\sqrt[3]{\frac{4(1-\ell^2)}{\ell^4}}$

– tedious, but doable

- Alternatively, convert the problem to semi-definite programming, then use solver such as SeDuMi

– [Nie (09)]

DWH Algorithm

$$X_{k+1} = X_k(a_k I + b_k X_k^* X_k)(I + c_k X_k^* X_k)^{-1}, \quad X_0 = A/\alpha$$

- Estimate α, ℓ_0 such that
  1. $\alpha \simeq \|A\|_2$,
  2. $\ell_0 \lesssim \sigma_{\min}(A/\alpha)$

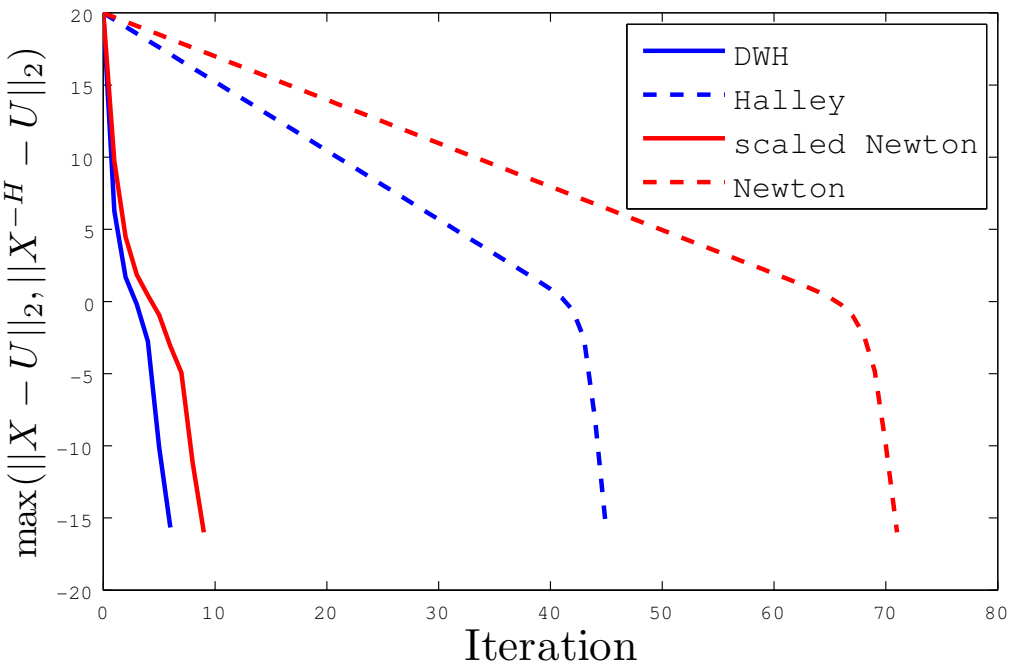
- $a_k = h(\ell_k), \quad b_k = (a_k - 1)^2/4, \quad c_k = a_k + b_k - 1,$

  where $h(\ell) = \sqrt{1+d} + \frac{1}{2}\sqrt{8 - 4d + \frac{8(2-\ell^2)}{\ell^2\sqrt{1+d}}}, \quad d = \frac{2^{2/3}(1-\ell^2)^{1/3}}{\ell^{4/3}}$

- Update: $\ell_{k+1} = \ell_k \dfrac{a_k + b_k \ell_k^2}{1 + c_k \ell_k^2}$

Efficiency of dynamical weighting

Cond. number		2	10	10^2	10^5	10^{10}	10^{15}	10^{20}
iter	DWH	3	4	4	5	5	6	6
	Halley	4	5	7	14	24	35	45
	scaled Newton	5	6	7	7	8	9	9



QR-based implementation of DWH

- Lemma:

$$Q_1 Q_2^* = A(I + A^*A)^{-1}, \text{ where } \begin{bmatrix} A \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \mathrm{R}$$

- DWH

$$\begin{aligned} X_{k+1} &= X_k(a_k I + X_k^* X_k)(b_k I + c_k X_k^* X_k)^{-1} \\ &= \frac{b_k}{c_k} X_k + \left(a_k - \frac{b_k}{c_k}\right) X_k (I + c_k X_k^* X_k)^{-1}, \end{aligned}$$

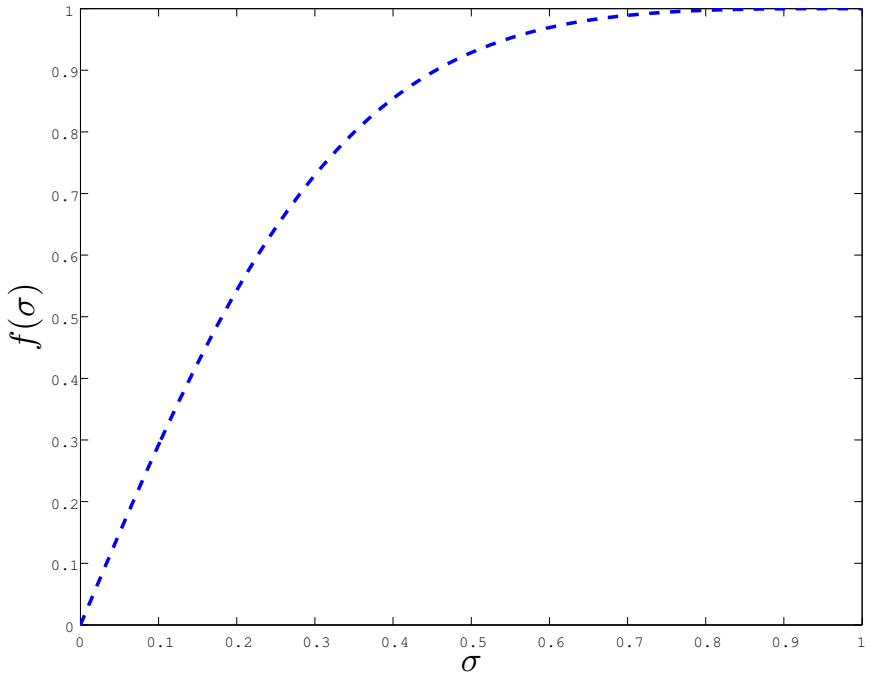
- QR-based DWH

$$\begin{cases} \begin{bmatrix} \sqrt{c_k} X_k \\ I \end{bmatrix} &= \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \\ X_{k+1} &= \frac{b_k}{c_k} X_k + \frac{1}{\sqrt{c_k}} \left(a_k - \frac{b_k}{c_k}\right) Q_1 Q_2^* \end{cases}$$

Halley vs. Dynamical weighted Halley

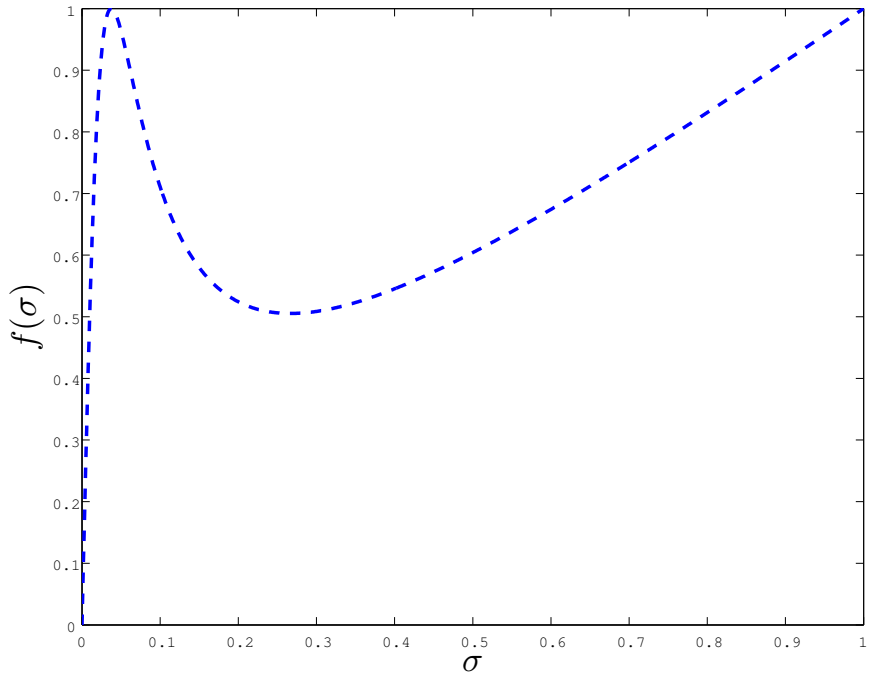
Halley: static parameters

it=0



DWH: dynamical parameters

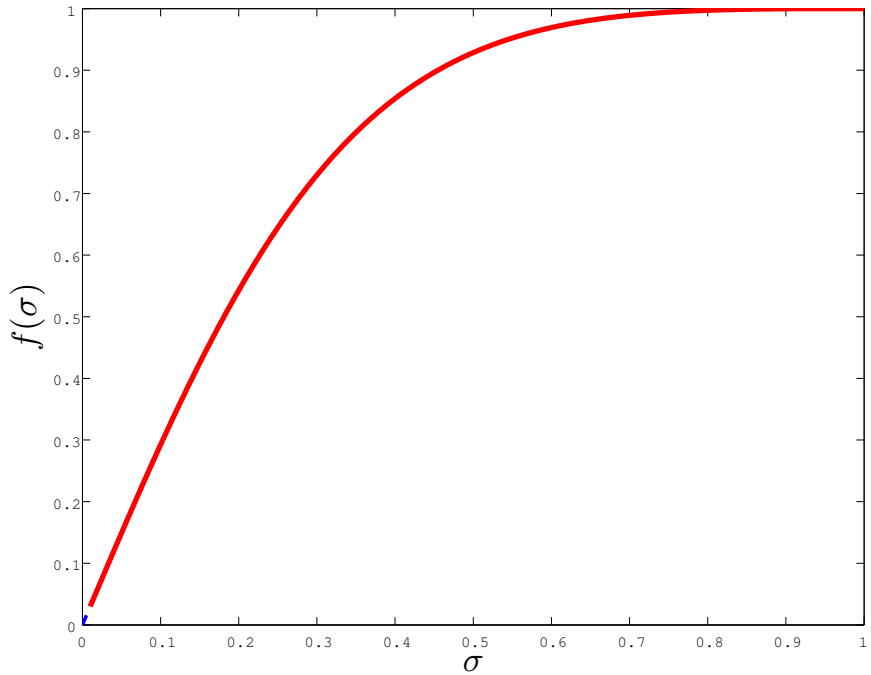
it=0



Halley vs. Dynamical weighted Halley

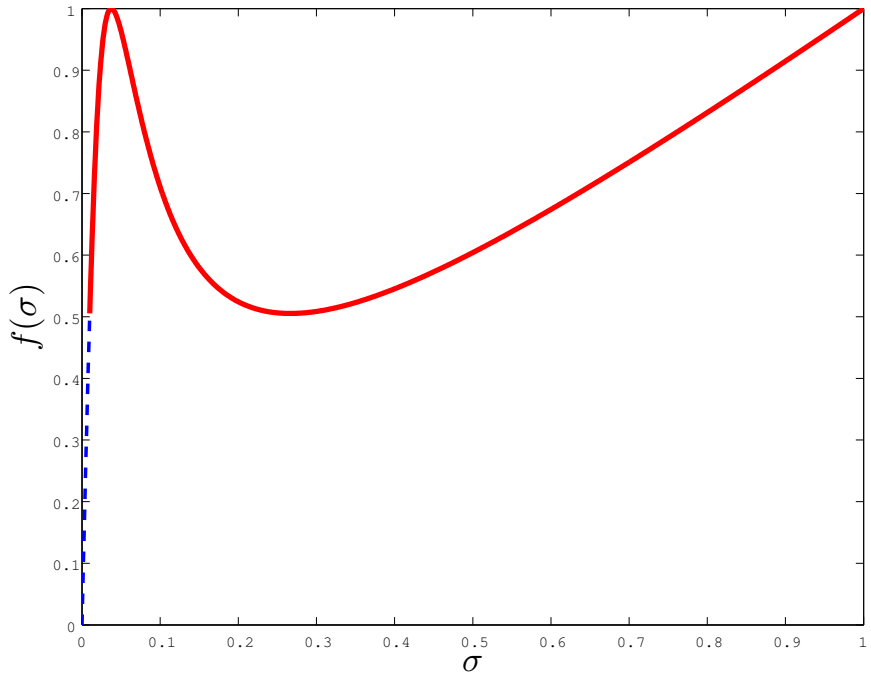
Halley: static parameters

it=0



DWH: dynamical parameters

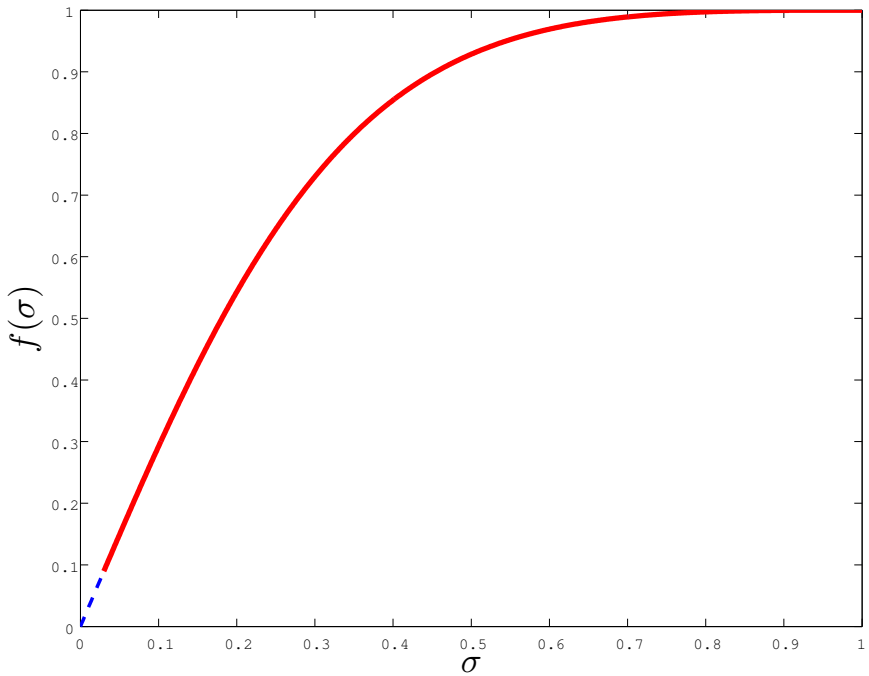
it=0



Halley vs. Dynamical weighted Halley

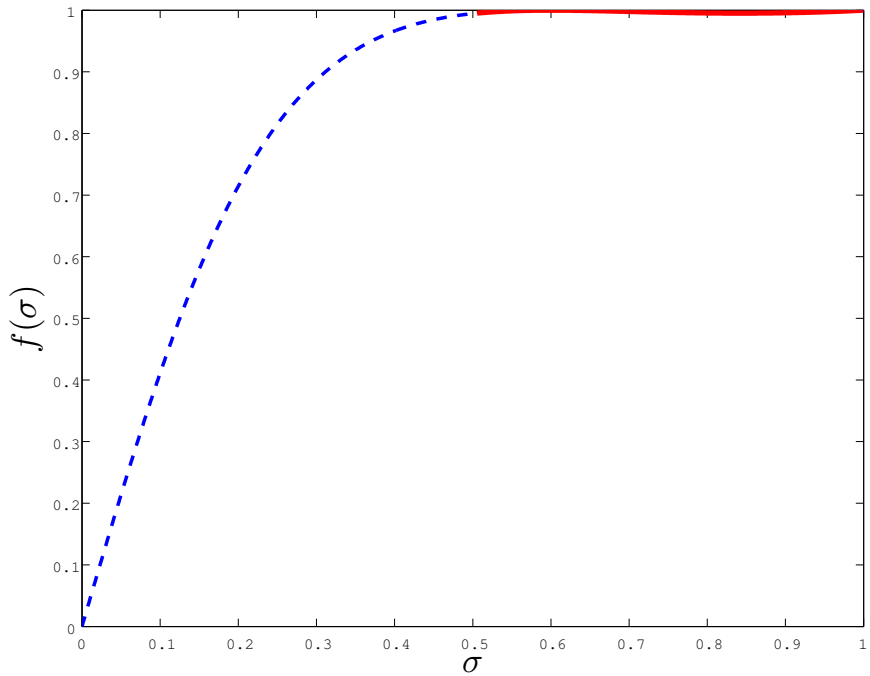
Halley: static parameters

it=1



DWH: dynamical parameters

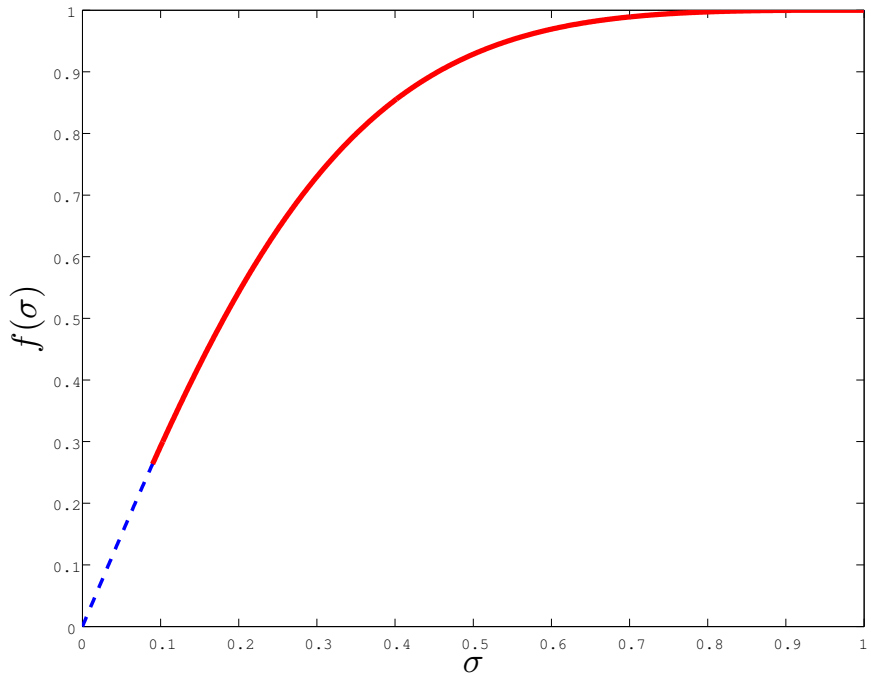
it=1



Halley vs. Dynamical weighted Halley

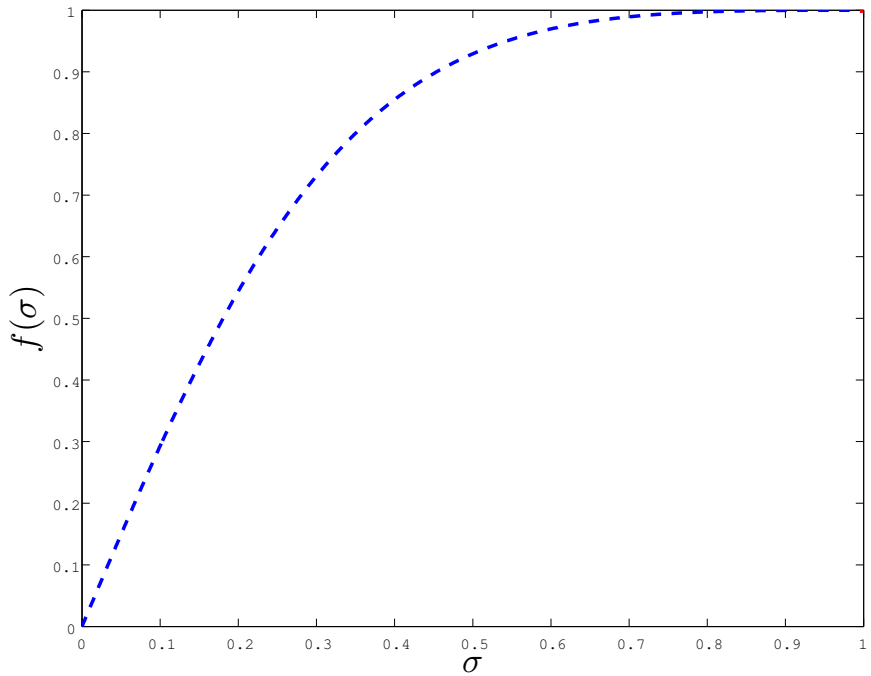
Halley: static parameters

it=2



DWH: dynamical parameters

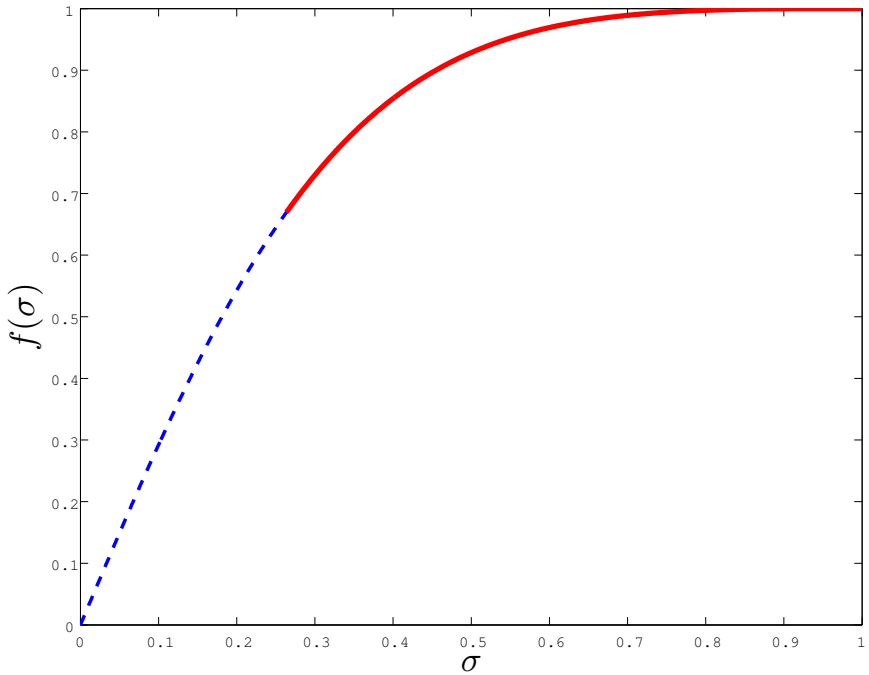
it=2



Halley vs. Dynamical weighted Halley

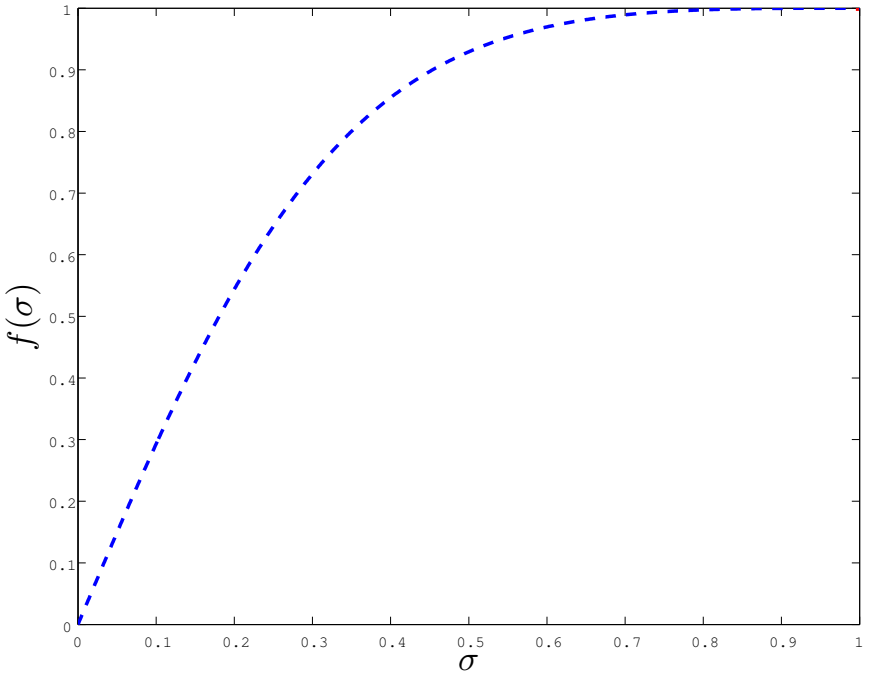
Halley: static parameters

it=3



DWH: dynamical parameters

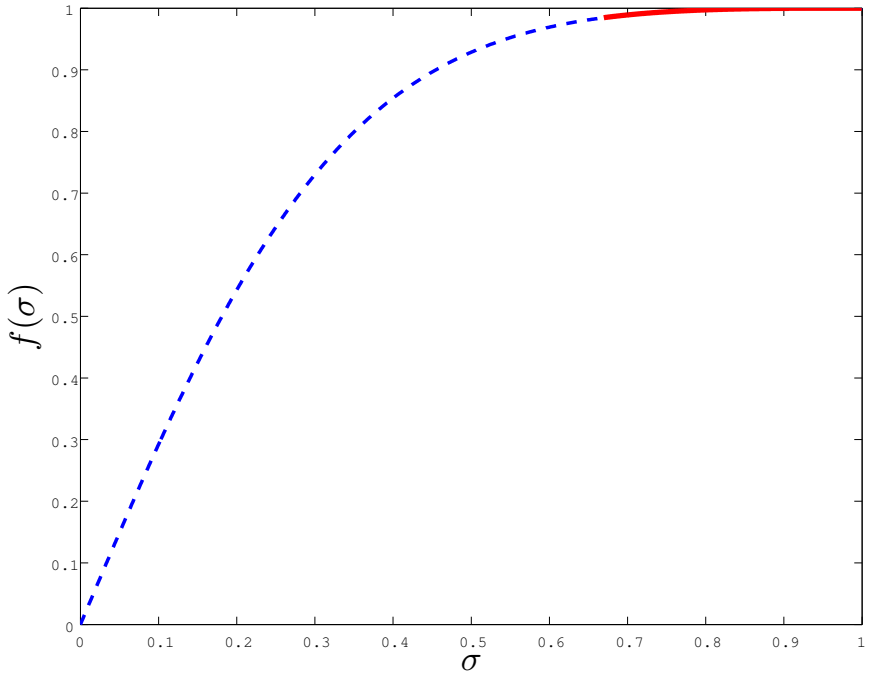
it=2



Halley vs. Dynamical weighted Halley

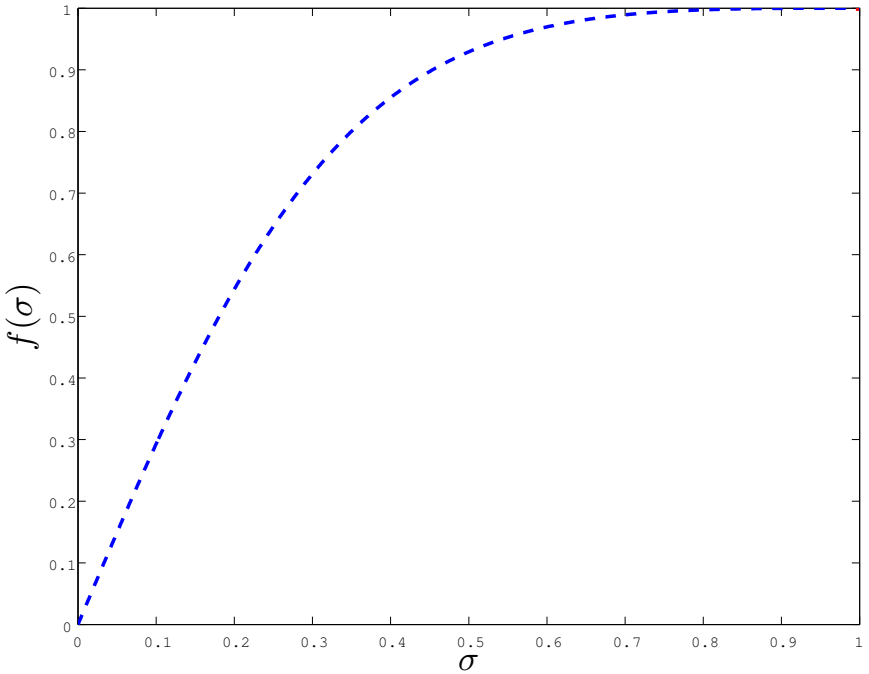
Halley: static parameters

it=4



DWH: dynamical parameters

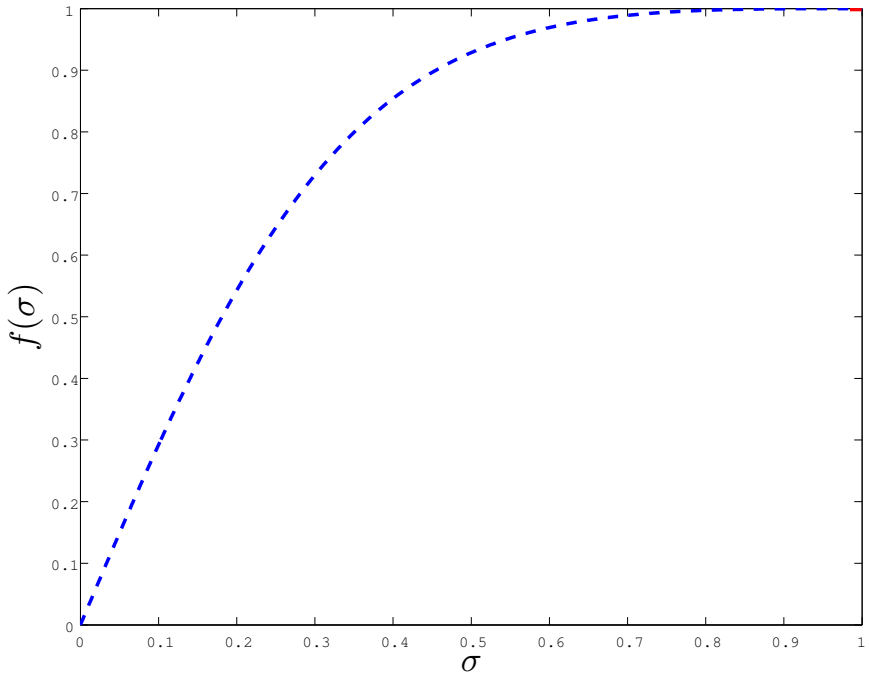
it=2



Halley vs. Dynamical weighted Halley

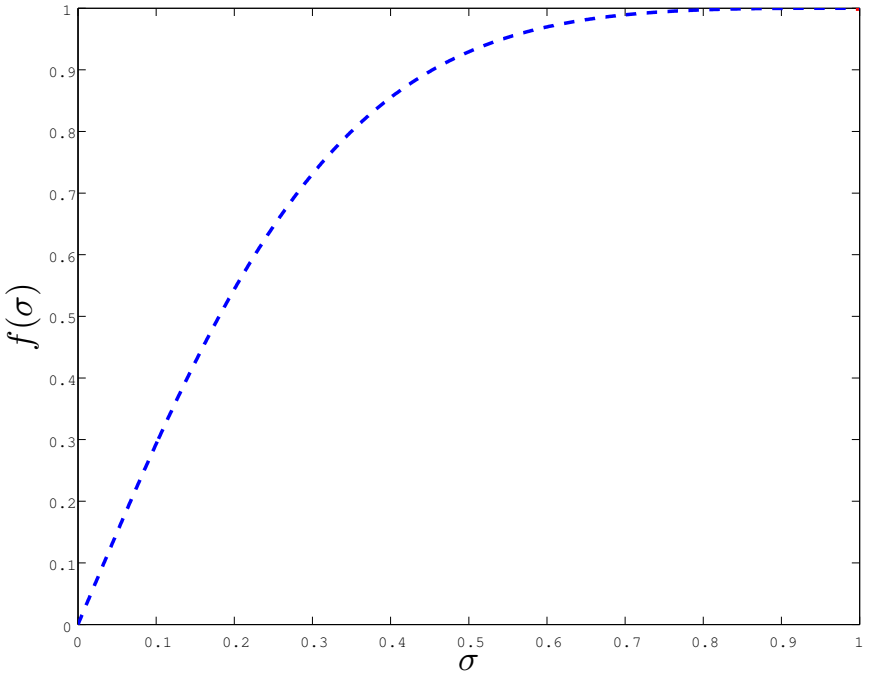
Halley: static parameters

it=5



DWH: dynamical parameters

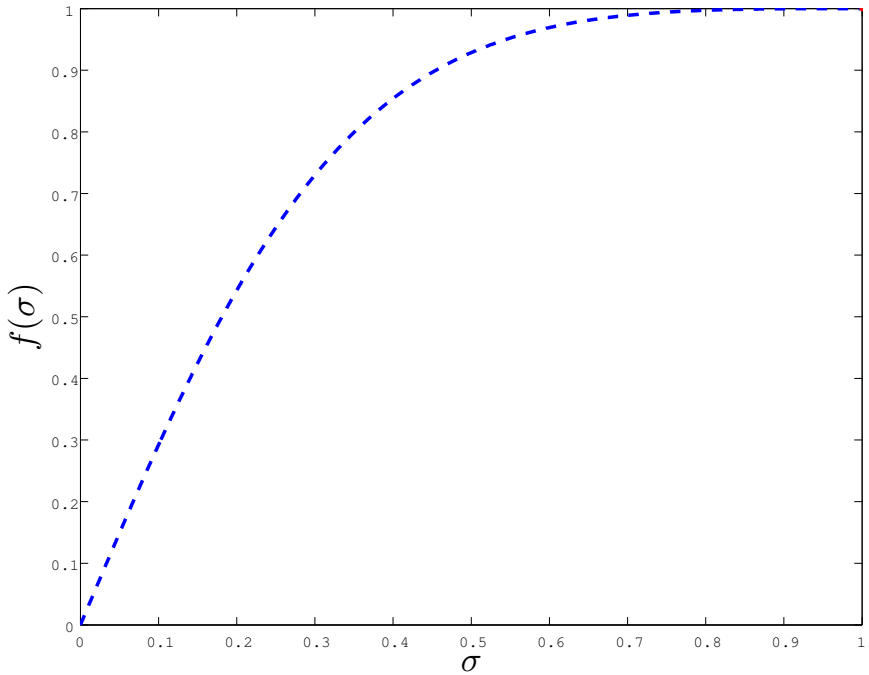
it=2



Halley vs. Dynamical weighted Halley

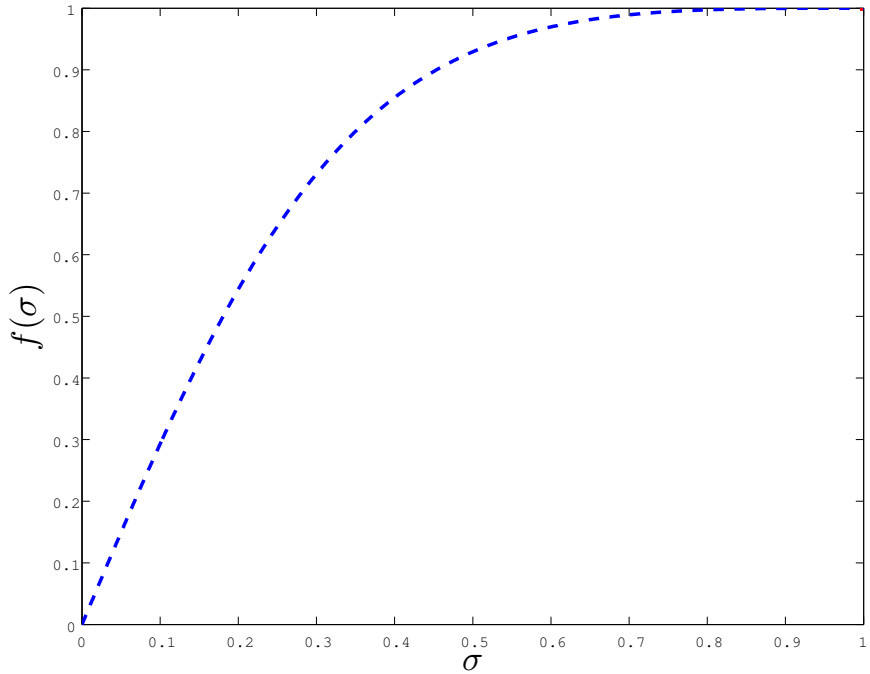
Halley: static parameters

it=6



DWH: dynamical parameters

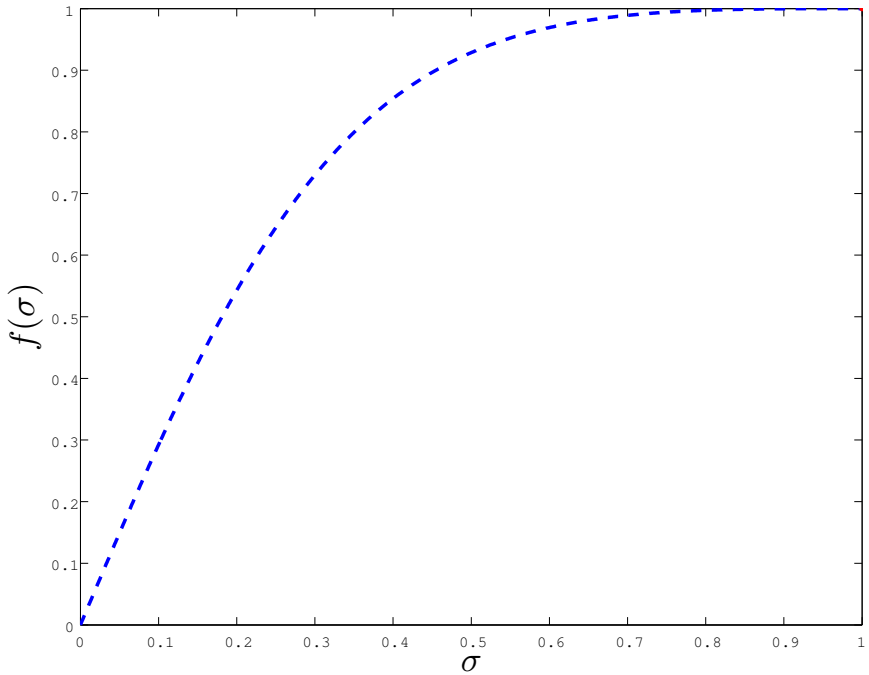
it=2



Halley vs. Dynamical weighted Halley

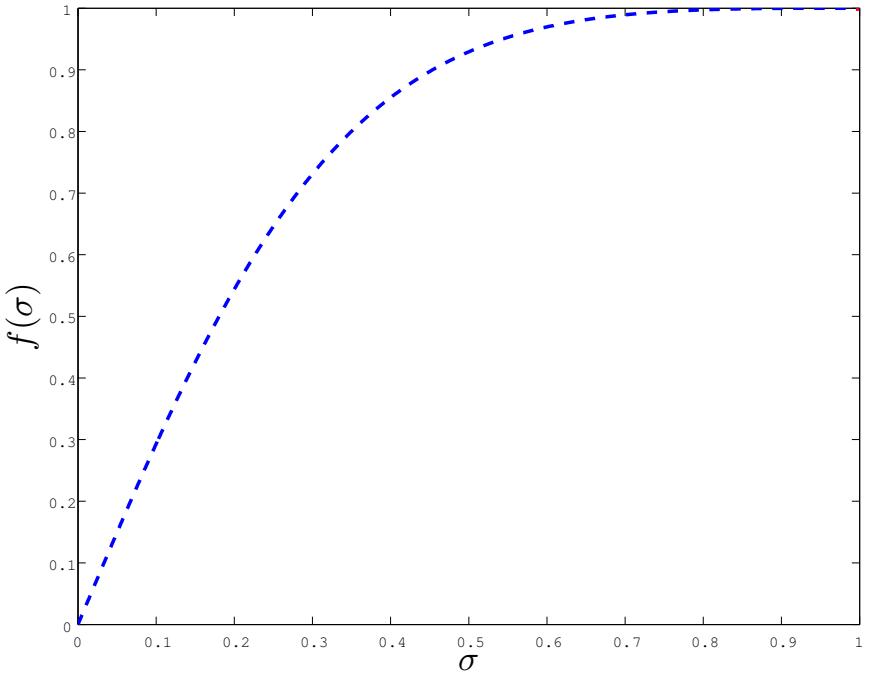
Halley: static parameters

it=6



DWH: dynamical parameters

it=2



dynamical parameters speed up convergence significantly

LAPACK SVD experiments, random matrices

Runtime(s)

	$n = 1000$	$n = 2000$	$n = 3000$	$n = 4000$
QDWH-SVD	6.4	40.9	127	286
Divide-Conquer	2.4	17.1	57.1	133
QR	5.4	40.0	126	287
Jacobi	7.8	60.4	248	579

Backward error $||A - \widehat{U}\widehat{\Sigma}\widehat{V}^T||_F / ||A||_F$

	$n = 1000$	$n = 2000$	$n = 3000$	$n = 4000$
QDWH-SVD	1.9e-15	2.0e-15	2.2e-15	2.3e-15
Divide-Conquer	4.5e-15	5.8e-15	7.0e-15	7.6e-15
QR	1.3e-14	1.8e-14	2.2e-14	2.6e-14
Jacobi	1.2e-14	1.8e-14	2.4e-14	3.0e-14

Distance from orthogonality $\max\{||\widehat{U}^T\widehat{U} - I||_F / \sqrt{n}, ||\widehat{V}^T\widehat{V} - I||_F / \sqrt{n}\}$.

	$n = 1000$	$n = 2000$	$n = 3000$	$n = 4000$
QDWH-SVD	7.4e-16	7.7e-16	7.9e-16	8.0e-16
Divide-Conquer	4.5e-15	5.8e-15	7.0e-15	7.6e-15
QR	1.3e-14	1.8e-14	2.2e-14	2.6e-14
Jacobi	1.2e-14	1.8e-14	2.4e-14	3.0e-14