Enhancing the PRIMME Eigensolver for Computing Accurately Singular Triplets of Large Matrices

Lingfei Wu and Andreas Stathopoulos

Department of Computer Science
College of William and Mary

April 10th, 2014

Introduction: applications of SVD

- Social network analysis: voting similarities among politicians

Introduction: applications of SVD

- Social network analysis: voting similarities among politicians
- Image-processing: calculation of Eigenfaces in face recognition

Introduction: applications of SVD

- Social network analysis: voting similarities among politicians
- Image-processing: calculation of Eigenfaces in face recognition
- Textual database searching: Google, Yahoo, and Baidu

Introduction: applications of SVD

- Social network analysis: voting similarities among politicians
- Image-processing: calculation of Eigenfaces in face recognition
- Textual database searching: Google, Yahoo, and Baidu
- Numerical linear algebra: least square fitting, rank, low-rank approximation, computation of pseudospectrum

Introduction: applications of SVD

- Social network analysis: voting similarities among politicians
- Image-processing: calculation of Eigenfaces in face recognition
- Textual database searching: Google, Yahoo, and Baidu
- Numerical linear algebra: least square fitting, rank, low-rank approximation, computation of pseudospectrum
- Variance reduction in Monte Carlo method

Introduction: what is SVD ?

Assume $A \in \Re^{m \times n}$ is a large, sparse matrix:

$$A = U \Sigma V^T$$
$$U^T U = I, V^T V = I, \Sigma = Diag$$

Introduction: what is SVD ?

Assume $A \in \Re^{m \times n}$ is a large, sparse matrix:

$$A = U \Sigma V^T$$
$$U^T U = I, V^T V = I, \Sigma = Diag$$

Our Problem: find *k* smallest singular values and corresponding left and right singular vectors of A

$$A v_i = \sigma_i u_i, \sigma_1 \leq \ldots \leq \sigma_k$$

Introduction: how to compute SVD ?

- A Hermitian eigenvalue problem on

Introduction: how to compute SVD ?

- A Hermitian eigenvalue problem on
  - Normal equations matrix $C = A^T A$ or $C = AA^T$
  - Augmented matrix $B = \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$

Introduction: how to compute SVD ?

- A Hermitian eigenvalue problem on
  - Normal equations matrix $C = A^T A$ or $C = AA^T$
  - Augmented matrix $B = \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$
- Lanczos bidiagonalization method (LBD)

Introduction: how to compute SVD ?

- A Hermitian eigenvalue problem on
  - Normal equations matrix $C = A^T A$ or $C = AA^T$
  - Augmented matrix $B = \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$
- Lanczos bidiagonalization method (LBD)

$$A = PB_dQ^T$$

$$B_d = X\Sigma Y^T$$

Where $U = PX$ and $V = QY$

Introduction: difference between methods

- **Convergence speed**

Introduction: difference between methods

- **Convergence speed**

Eigen methods on C

- fast for largest SVs
- slow for smallest SVs

Introduction: difference between methods

- **Convergence speed**

Eigen methods on C	○ fast for largest SVs ○ slow for smallest SVs
Eigen methods on B	○ slower for largest SVs ○ extremely slow for smallest SVs (interior eigenvalue problem)

Introduction: difference between methods

- **Convergence speed**

Eigen methods on C	◦ fast for largest SVs ◦ slow for smallest SVs
Eigen methods on B	◦ slower for largest SVs ◦ extremely slow for smallest SVs (interior eigenvalue problem)
LBD on A	◦ fast for largest SVs ◦ similar to C but exhibits irregular convergence for smallest SVs

Introduction: difference between methods

- **Accuracy**

Introduction: difference between methods

- **Accuracy**

Eigen methods on C	◦ can only achieve accuracy of $O(\kappa(A)\|A\|\epsilon_{mach})$

Introduction: difference between methods

- **Accuracy**

Eigen methods on C	◦ can only achieve accuracy of $O(\kappa(A)\|A\|\epsilon_{mach})$
Eigen methods on B	◦ can achieve accuracy of $O(\|A\|\epsilon_{mach})$

Introduction: difference between methods

- **Accuracy**

Eigen methods on C	can only achieve accuracy of $O(\kappa(A)\|A\|\epsilon_{mach})$
Eigen methods on B	can achieve accuracy of $O(\|A\|\epsilon_{mach})$
LBD on A	can achieve accuracy of $O(\|A\|\epsilon_{mach})$

Related work: LBD

- Advantages of LBD:

Related work: LBD

- Advantages of LBD:
  - Builds same space as Lanczos on C but avoids numerical issues by working on A

Related work: LBD

- Advantages of LBD:
  - Builds same space as Lanczos on C but avoids numerical issues by working on A
  - Computationally inexpensive to combine with the harmonic and refined projection methods

Related work: LBD

- Advantages of LBD:
  - Builds same space as Lanczos on C but avoids numerical issues by working on A
  - Computationally inexpensive to combine with the harmonic and refined projection methods
  - Inherits the global convergence of Lanczos when seeking many singular triplets

Related work: LBD

- Advantages of LBD:
  - Builds same space as Lanczos on C but avoids numerical issues by working on A
  - Computationally inexpensive to combine with the harmonic and refined projection methods
  - Inherits the global convergence of Lanczos when seeking many singular triplets
- Drawbacks of LBD:

Related work: LBD

- Advantages of LBD:
  - Builds same space as Lanczos on C but avoids numerical issues by working on A
  - Computationally inexpensive to combine with the harmonic and refined projection methods
  - Inherits the global convergence of Lanczos when seeking many singular triplets
- Drawbacks of LBD:
  - Orthogonality loss, large memory demands and irregular convergence

Related work: LBD

- Advantages of LBD:
  - Builds same space as Lanczos on C but avoids numerical issues by working on A
  - Computationally inexpensive to combine with the harmonic and refined projection methods
  - Inherits the global convergence of Lanczos when seeking many singular triplets
- Drawbacks of LBD:
  - Orthogonality loss, large memory demands and irregular convergence
  - Current SVD solvers not reflect remarkable algorithmic progress

Related work: LBD

- Advantages of LBD:
  - Builds same space as Lanczos on C but avoids numerical issues by working on A
  - Computationally inexpensive to combine with the harmonic and refined projection methods
  - Inherits the global convergence of Lanczos when seeking many singular triplets
- Drawbacks of LBD:
  - Orthogonality loss, large memory demands and irregular convergence
  - Current SVD solvers not reflect remarkable algorithmic progress
  - Cannot use preconditioning to accelerate convergence

Related work: the JDSVD method

- Advantages of the JDSVD method:

Related work: the JDSVD method

- Advantages of the JDSVD method:
  - Can take advantage of preconditioning

Related work: the JDSVD method

- Advantages of the JDSVD method:
  - Can take advantage of preconditioning
  - Two search spaces share advantages of LBD

Related work: the JDSVD method

- Advantages of the JDSVD method:
  - Can take advantage of preconditioning
  - Two search spaces share advantages of LBD
  - No numerical accuracy problem

Related work: the JDSVD method

- Advantages of the JDSVD method:
  - Can take advantage of preconditioning
  - Two search spaces share advantages of LBD
  - No numerical accuracy problem
- Drawbacks of the JDSVD method:

Related work: the JDSVD method

- Advantages of the JDSVD method:
  - Can take advantage of preconditioning
  - Two search spaces share advantages of LBD
  - No numerical accuracy problem
- Drawbacks of the JDSVD method:
  - Correction equation working on B may not be efficient

Related work: the JDSVD method

- Advantages of the JDSVD method:
  - Can take advantage of preconditioning
  - Two search spaces share advantages of LBD
  - No numerical accuracy problem
- Drawbacks of the JDSVD method:
  - Correction equation working on B may not be efficient
  - Still in development and only MATLAB implementation

Motivation I: our goal for an SVD solver

Extremely challenging task for small SVs:

- large sparse matrix $\Rightarrow$ No shift-and-invert

Motivation I: our goal for an SVD solver

Extremely challenging task for small SVs:

- large sparse matrix $\Rightarrow$ No shift-and-invert
- very slow convergence $\Rightarrow$ restarting and preconditioning

Motivation I: our goal for an SVD solver

Extremely challenging task for small SVs:

- large sparse matrix $\Rightarrow$ No shift-and-invert
- very slow convergence $\Rightarrow$ restarting and preconditioning
- very few SVD solvers:
  - SVDPACK: Lanczos and trace-minimization methods working on B or C for only largest SVs
  - PROPACK: LBD for largest SVs, using shift-and-inverting for smallest SVs

Motivation I: our goal for an SVD solver

Extremely challenging task for small SVs:

- large sparse matrix $\Rightarrow$ No shift-and-invert
- very slow convergence $\Rightarrow$ restarting and preconditioning
- very few SVD solvers:
  - SVDPACK: Lanczos and trace-minimization methods working on B or C for only largest SVs
  - PROPACK: LBD for largest SVs, using shift-and-inverting for smallest SVs

$\Rightarrow$ calls for full functionality, highly-optimized SVD solver

Motivation I: our goal for an SVD solver

Extremely challenging task for small SVs:

- large sparse matrix ⇒ No shift-and-invert
- very slow convergence ⇒ restarting and preconditioning
- very few SVD solvers:
  - SVDPACK: Lanczos and trace-minimization methods working on B or C for only largest SVs
  - PROPACK: LBD for largest SVs, using shift-and-inverting for smallest SVs

⇒ calls for full functionality, highly-optimized SVD solver

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

Motivation II: building an SVD solver on PRIMME

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

- Over 12 eigenmethods including near optimal GD+k and JDQMR methods

Motivation II: building an SVD solver on PRIMME

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

- Over 12 eigenmethods including near optimal GD+k and JDQMR methods
- Supports seeking interior eigenvalues

Motivation II: building an SVD solver on PRIMME

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

- Over 12 eigenmethods including near optimal GD+k and JDQMR methods
- Supports seeking interior eigenvalues
- Accepts initial guesses for all required eigenvectors

Motivation II: building an SVD solver on PRIMME

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

- Over 12 eigenmethods including near optimal GD+k and JDQMR methods
- Supports seeking interior eigenvalues
- Accepts initial guesses for all required eigenvectors
- Accepts many shifts and finds the closest eigenvalue to each shift

Motivation II: building an SVD solver on PRIMME

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

- Over 12 eigenmethods including near optimal GD+k and JDQMR methods
- Supports seeking interior eigenvalues
- Accepts initial guesses for all required eigenvectors
- Accepts many shifts and finds the closest eigenvalue to each shift
- Accepts preconditioner for C or B, or if $M \approx A^{-1}$, uses $MM^T \approx C^{-1}$ and $\begin{bmatrix} 0 & M \\ M^T & 0 \end{bmatrix} \approx B^{-1}$

Motivation II: building an SVD solver on PRIMME

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

- Over 12 eigenmethods including near optimal GD+k and JDQMR methods
- Supports seeking interior eigenvalues
- Accepts initial guesses for all required eigenvectors
- Accepts many shifts and finds the closest eigenvalue to each shift
- Accepts preconditioner for C or B, or if $M \approx A^{-1}$, uses $MM^T \approx C^{-1}$ and $\begin{bmatrix} 0 & M \\ M^T & 0 \end{bmatrix} \approx B^{-1}$
- A robust framework: subspace acceleration, locking mechanism

Motivation II: building an SVD solver on PRIMME

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

- Over 12 eigenmethods including near optimal GD+k and JDQMR methods
- Supports seeking interior eigenvalues
- Accepts initial guesses for all required eigenvectors
- Accepts many shifts and finds the closest eigenvalue to each shift
- Accepts preconditioner for C or B, or if $M \approx A^{-1}$, uses $MM^T \approx C^{-1}$ and $\begin{bmatrix} 0 & M \\ M^T & 0 \end{bmatrix} \approx B^{-1}$
- A robust framework: subspace acceleration, locking mechanism
- Parallel, high performance implementation for large, sparse, hermitian matrices

Motivation III: compare search spaces

If u_1, v_1 are left and right initial guesses, each method builds Krylov space:

- Eigen methods on C: $K_k(A^T A, v_1)$

Motivation III: compare search spaces

If u_1, v_1 are left and right initial guesses, each method builds Krylov space:

- Eigen methods on C:
  $K_k(A^TA, v_1)$
- LBD on A and A^T:
  $K_k(AA^T, Av_1)$, $K_k(A^TA, v_1)$

Motivation III: compare search spaces

If u_1, v_1 are left and right initial guesses, each method builds Krylov space:

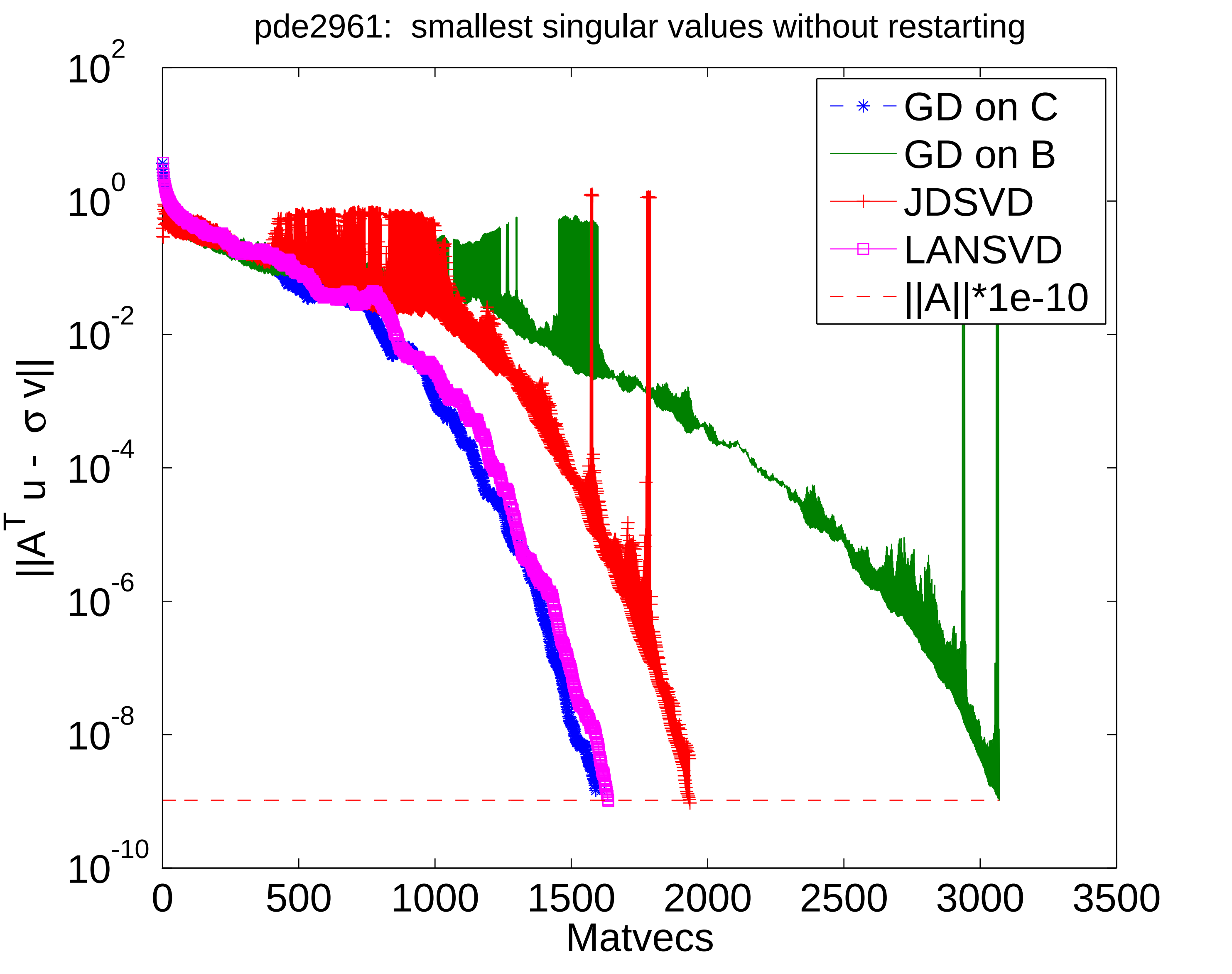
- Eigen methods on C:
  $K_k(A^T A, v_1)$
- LBD on A and A^T:
  $K_k(AA^T, Av_1)$, $K_k(A^T A, v_1)$
- Eigen methods on B:
  $$\begin{pmatrix} K_{\frac{k}{2}}(AA^T, u_1) \\ K_{\frac{k}{2}}(A^T A, v_1) \end{pmatrix} \oplus \begin{pmatrix} K_{\frac{k}{2}}(AA^T, Av_1) \\ K_{\frac{k}{2}}(A^T A, A^T u_1) \end{pmatrix}$$

Motivation III: compare search spaces

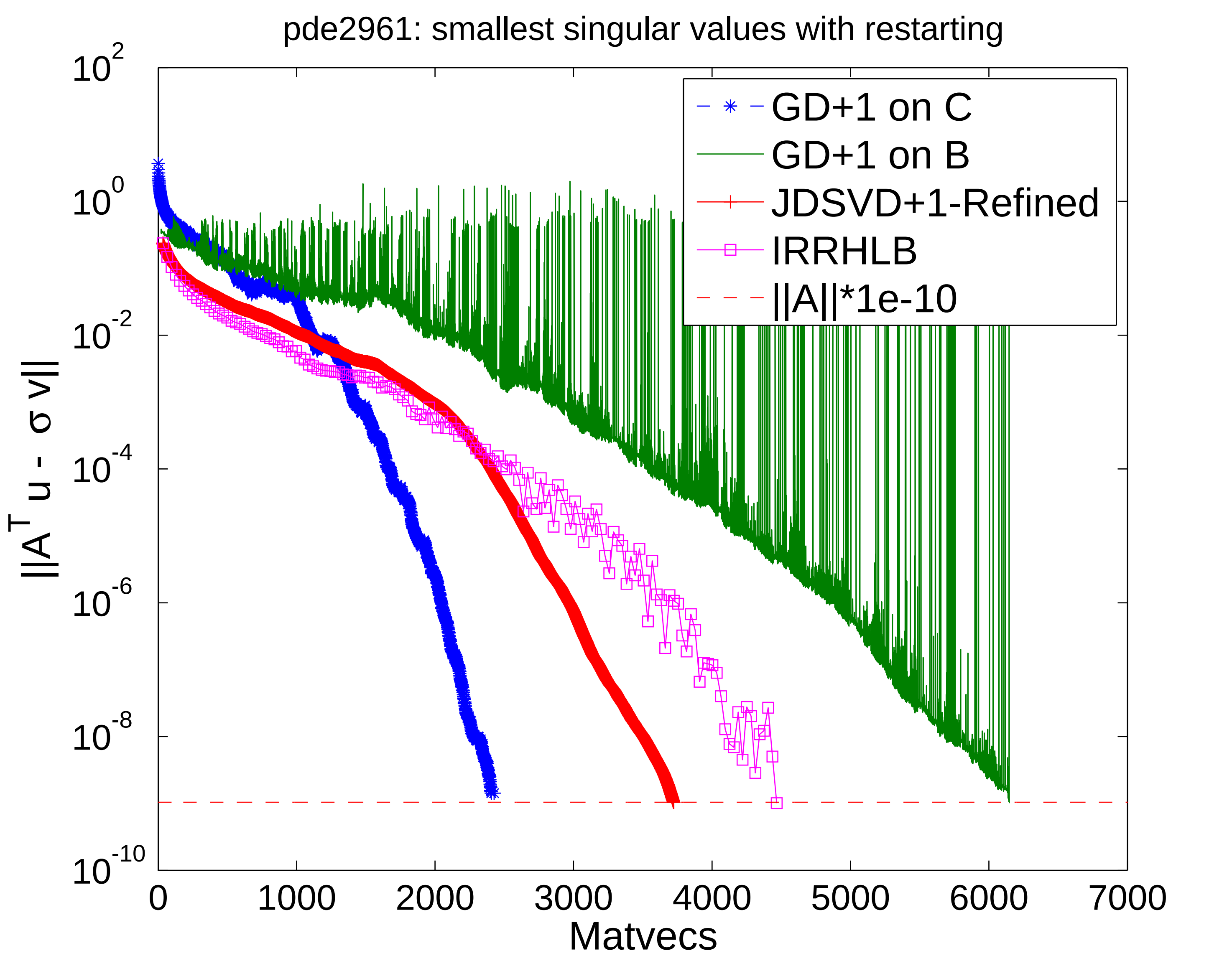
If u_1, v_1 are left and right initial guesses, each method builds Krylov space:

- Eigen methods on C:
  $K_k(A^TA, v_1)$
- LBD on A and A^T:
  $K_k(AA^T, Av_1)$, $K_k(A^TA, v_1)$
- Eigen methods on B:
  $$\begin{pmatrix} K_{\frac{k}{2}}(AA^T, u_1) \\ K_{\frac{k}{2}}(A^TA, v_1) \end{pmatrix} \oplus \begin{pmatrix} K_{\frac{k}{2}}(AA^T, Av_1) \\ K_{\frac{k}{2}}(A^TA, A^Tu_1) \end{pmatrix}$$
- JDSVD method (outer iteration) on A and A^T:
  $K_{\frac{k}{2}}(AA^T, u_1) \oplus K_{\frac{k}{2}}(AA^T, Av_1)$,
  $K_{\frac{k}{2}}(A^TA, v_1) \oplus K_{\frac{k}{2}}(A^TA, A^Tu_1)$

Motivation III: compare search spaces



Motivation III: the impact of restarting



primme_svds: the two stage strategy

- GD+1 on C is best for a few smallest SVs, but limited by accuracy
- need another phase to refine the accuracy

primme_svds: the two stage strategy

- GD+1 on C is best for a few smallest SVs, but limited by accuracy
- need another phase to refine the accuracy

Our solution: a hybrid, two-stage singular value method

primme_svds: the two stage strategy

- GD+1 on C is best for a few smallest SVs, but limited by accuracy
- need another phase to refine the accuracy

Our solution: a hybrid, two-stage singular value method

* Stage I: works on C to max residual tolerance $\max\left(\sigma_i \delta_{user}\|A\|, \|A\|^2 \epsilon_{mach}\right)$
  - Must dynamically adjust tolerance in PRIMME to meet user tolerance

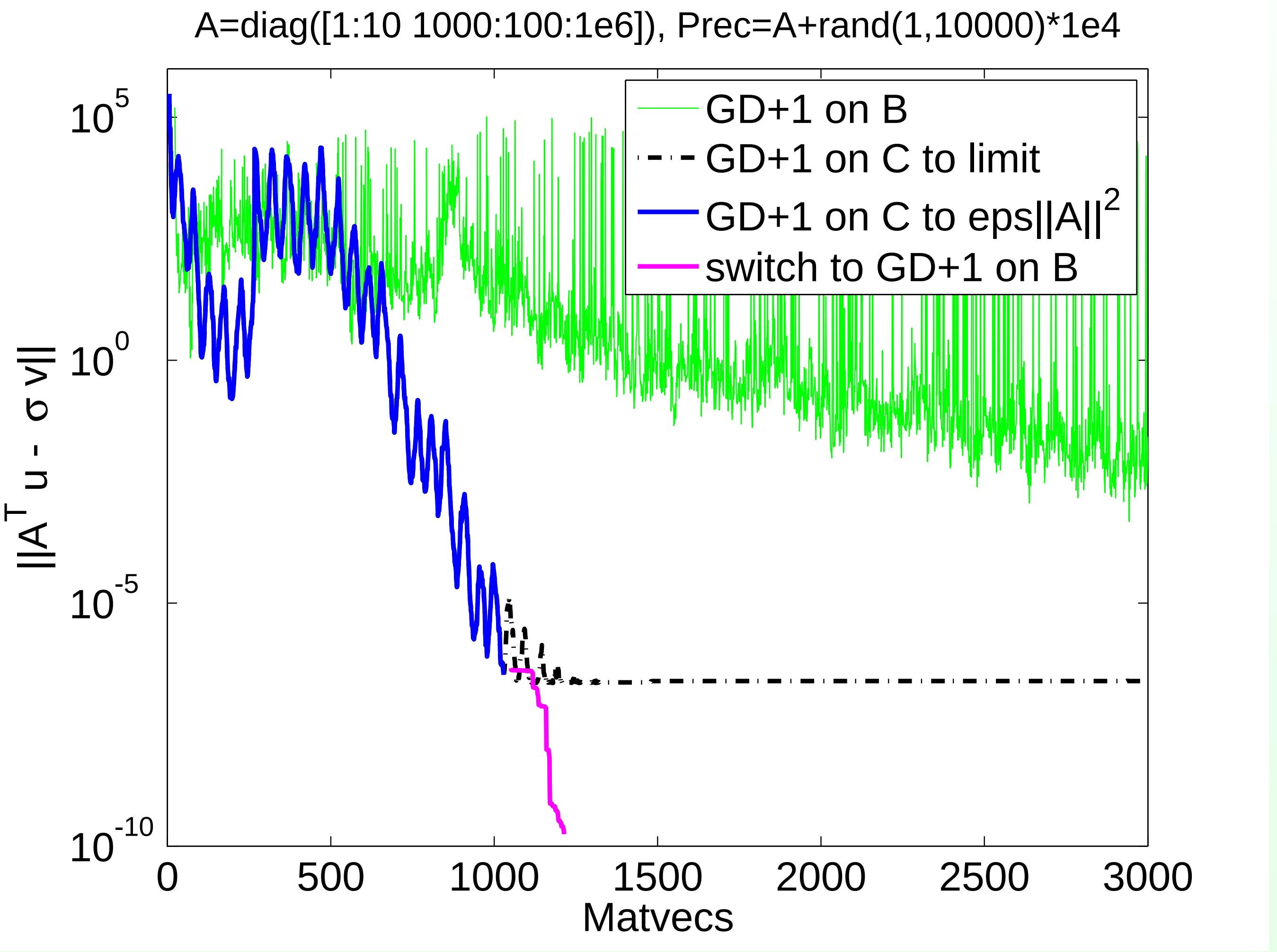
primme_svds: the two stage strategy

- GD+1 on C is best for a few smallest SVs, but limited by accuracy
- need another phase to refine the accuracy

Our solution: a hybrid, two-stage singular value method

* Stage I: works on C to max residual tolerance $\max\left(\sigma_i \delta_{user} \|A\|, \|A\|^2 \epsilon_{mach}\right)$
  - Must dynamically adjust tolerance in PRIMME to meet user tolerance
* Stage II: works on B to improve the approximations from C to user required tolerance $\delta_{user}\|A\|$

primme_svds: an example



Stage I of primme_svds: working on C

- What's the tolerance threshold to converge to?

Stage I of primme_svds: working on C

- What's the tolerance threshold to converge to?
- How to dynamically adjust the tolerance in PRIMME?

Stage I of primme_svds: working on C

Let $(\tilde{\sigma}, \tilde{u}, \tilde{v})$ be a targeted singular triplet of A

$$r_v = A\tilde{v} - \tilde{\sigma}\tilde{u}, \qquad r_u = A^T\tilde{u} - \tilde{\sigma}\tilde{v},$$
$$r_C = C\tilde{v} - \tilde{\sigma}^2\tilde{v}, \quad r_B = B\begin{bmatrix}\tilde{v}\\ \tilde{u}\end{bmatrix} - \tilde{\sigma}\begin{bmatrix}\tilde{v}\\ \tilde{u}\end{bmatrix}.$$

If $\|v_i\| = 1$, $\|u_i\| = \|Av_i/\sigma_i\| = 1$, then $r_v = 0$ and

$$\|r_u\| = \frac{\|r_C\|}{\tilde{\sigma}} = \|r_B\|\sqrt{2}$$

Thus, the stopping criterion for the methods on C becomes,

$$\delta_C = \max\left(\delta_{user}\,\tilde{\sigma}/\|A\|, \epsilon_{mach}\right)$$

Stage II of primme_svds: working on B

- **Inputs from $\mathbf{C}$:**
  - Accurate shifts for interior eigenvalue problem
  - Good initial guesses formed by eigenvectors from C

Stage II of primme_svds: working on B

- **Inputs from $\mathbf{C}$:**
  - Accurate shifts for interior eigenvalue problem
  - Good initial guesses formed by eigenvectors from C

$\Rightarrow$ Calls for JDQMR - one of near-optimal methods in PRIMME

Stage II of primme_svds: working on B

- **Inputs from C:**
  - Accurate shifts for interior eigenvalue problem
  - Good initial guesses formed by eigenvectors from C

  $\Rightarrow$ Calls for JDQMR - one of near-optimal methods in PRIMME
- **Irregular convergence of Rayleigh Ritz (RR) on B**

Stage II of primme_svds: working on B

- **Inputs from $\mathbf{C}$:**
  - Accurate shifts for interior eigenvalue problem
  - Good initial guesses formed by eigenvectors from C

  $\Rightarrow$ Calls for JDQMR - one of near-optimal methods in PRIMME
- **Irregular convergence of Rayleigh Ritz (RR) on $\mathbf{B}$**

  $\Rightarrow$ Enhance PRIMME with refined projection method

Stage II of primme_svds: working on B

Refined projection minimizes the residual $\|BVy - \tilde{\sigma}Vy\|$ where V search space for a given user shift $\tilde{\sigma}$

- QR factorization on $BV - \tilde{\sigma}V$ only after restart
- one column updating for Q and R during iteration
- computational cost similar with the RR method

Stage II of primme_svds: working on B

Eigenvalue method VS Iterative Refinement (IR)?

- correction equation on B equivalent to IR but JD leverages subspace acceleration with near-by eigenvectors
- stops the linear solver optimally
- IR may fail without deflation strategies

Outline of the implementation of primme_svds

- Developed PRIMME MEX, a MATLAB interface for PRIMME

Outline of the implementation of primme_svds

- Developed PRIMME MEX, a MATLAB interface for PRIMME
- User interfaces are similar to MATLAB eigs() and svds(), but allow access to full-functionality of PRIMME

Outline of the implementation of primme_svds

- Developed PRIMME MEX, a MATLAB interface for PRIMME
- User interfaces are similar to MATLAB eigs() and svds(), but allow access to full-functionality of PRIMME
- Refined projection implemented in PRIMME, and C implementation of primme_svds in PRIMME soon

Evaluation: Test matrices

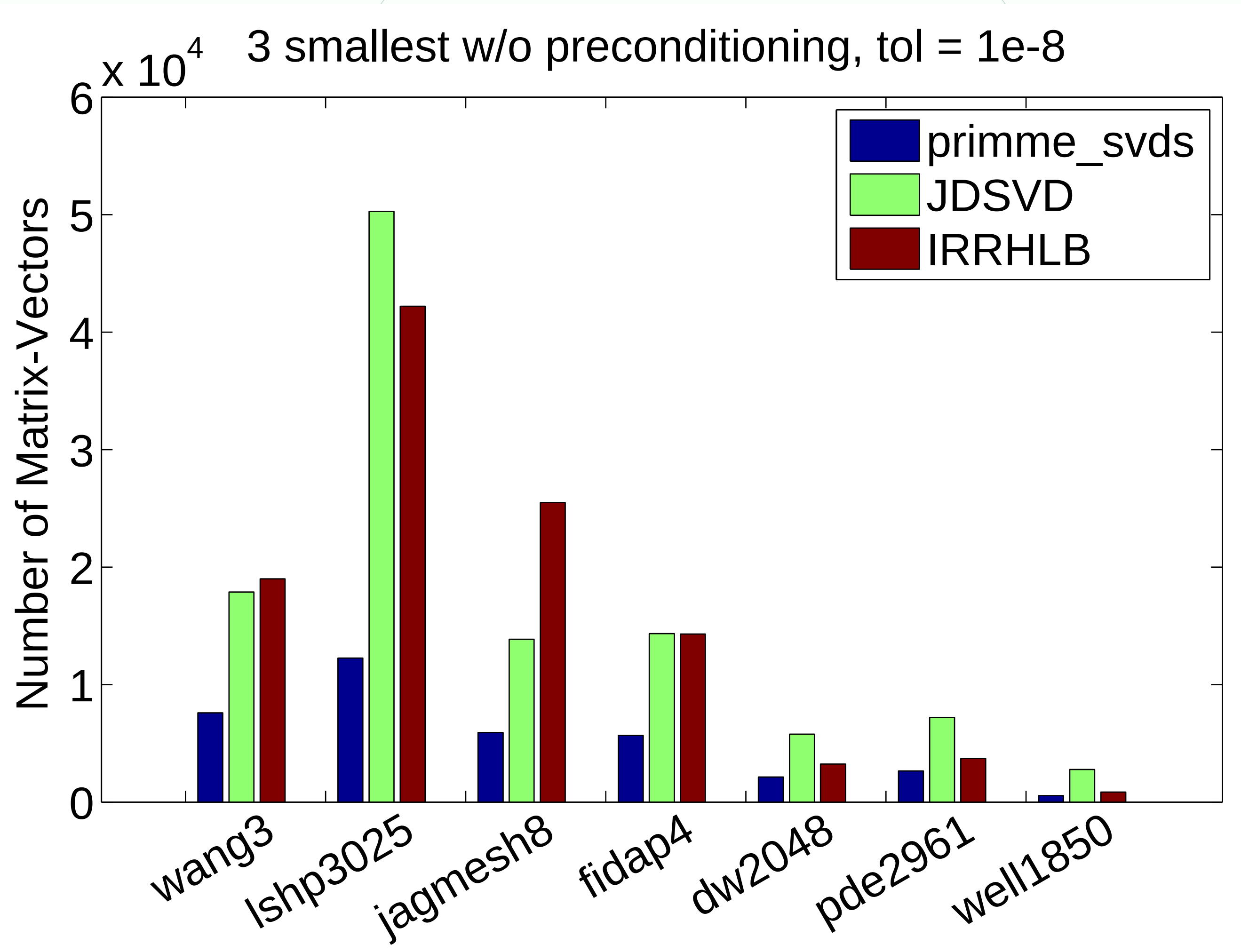
Table 1: Properties of the test matrices

Matrix	well1850	pde2961	dw2048	fidap4	jagmesh8	lshp3025	wang3
order	1850	2961	2048	1601	1141	3025	26064
$\kappa(A)$	1.1e2	9.5e2	5.3e3	5.2e3	5.9e4	2.2e5	1.1e4
$\|A\|_2$	1.8e0	1.0e1	1.0e0	1.6e0	6.8e0	7.0e0	2.7e-1
$gap_{min}(1)$	3.0e-3	8.2e-3	2.6e-3	1.5e-3	1.7e-3	1.8e-3	7.4e-5
$gap_{min}(3)$	3.0e-3	2.4e-3	2.9e-4	2.5e-4	1.6e-3	9.1e-4	1.9e-5
$gap_{min}(5)$	3.0e-3	2.4e-3	2.9e-4	2.5e-4	4.8e-5	1.8e-4	1.9e-5
$gap_{min}(10)$	2.6e-3	7.0e-4	1.6e-4	2.5e-4	4.8e-5	2.2e-5	6.6e-6

Other state-of-the-art methods to compare:

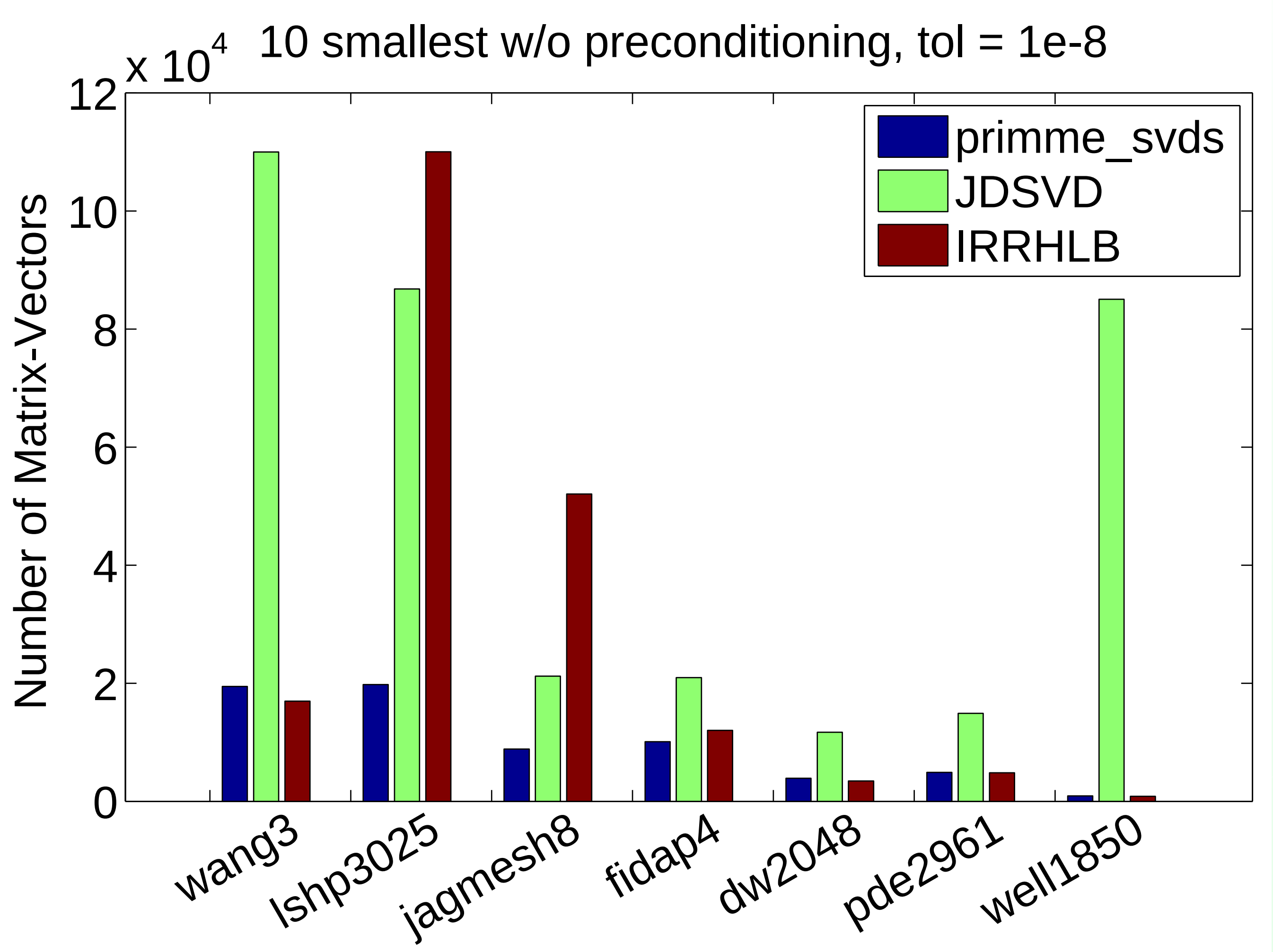
- JDSVD: (Hochstenbach, 2001)
- IRRHLB: (Jia, 2010)

Evaluation: Without preconditioning



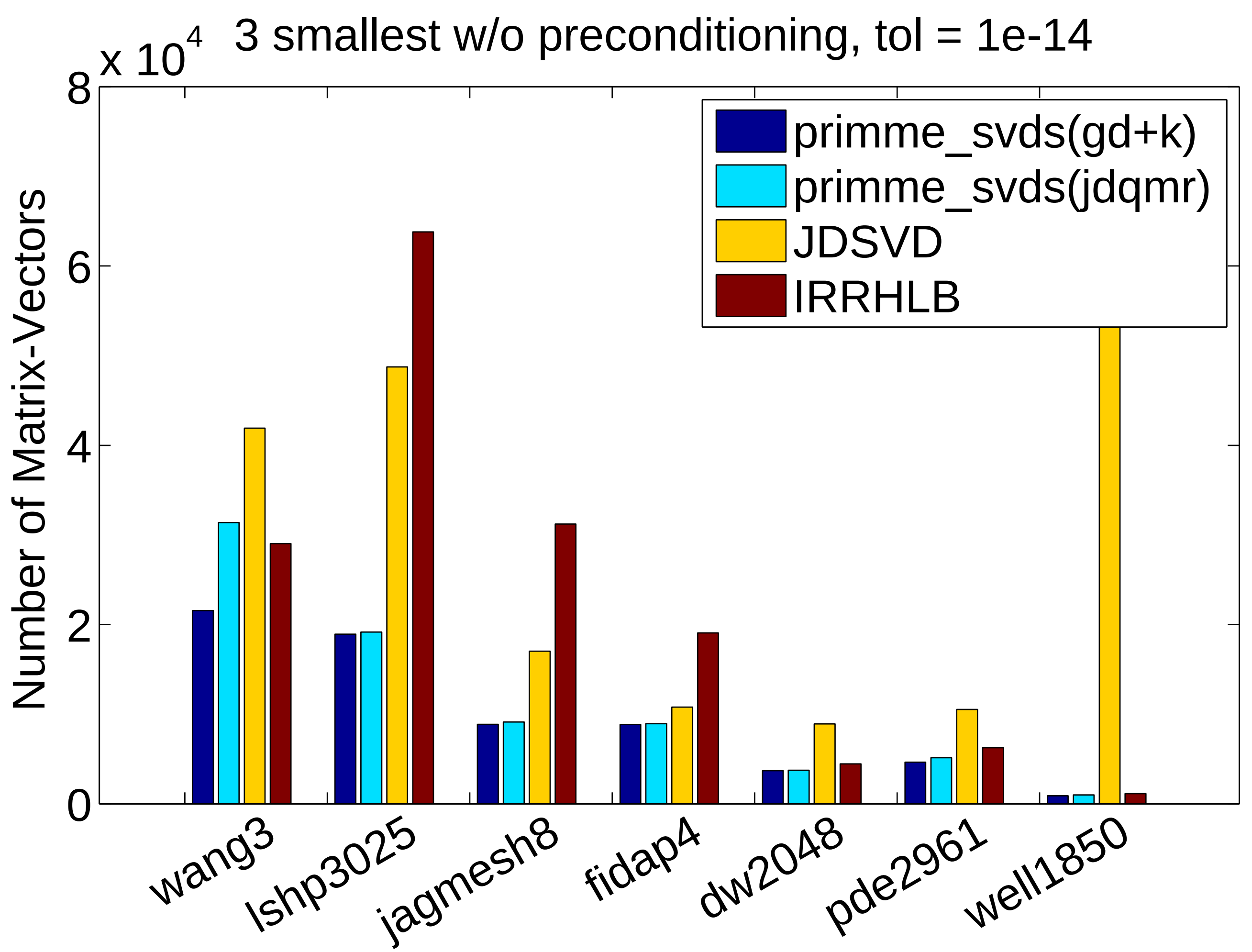
primme_svds (only first stage) is the fastest method

Evaluation: Without preconditioning



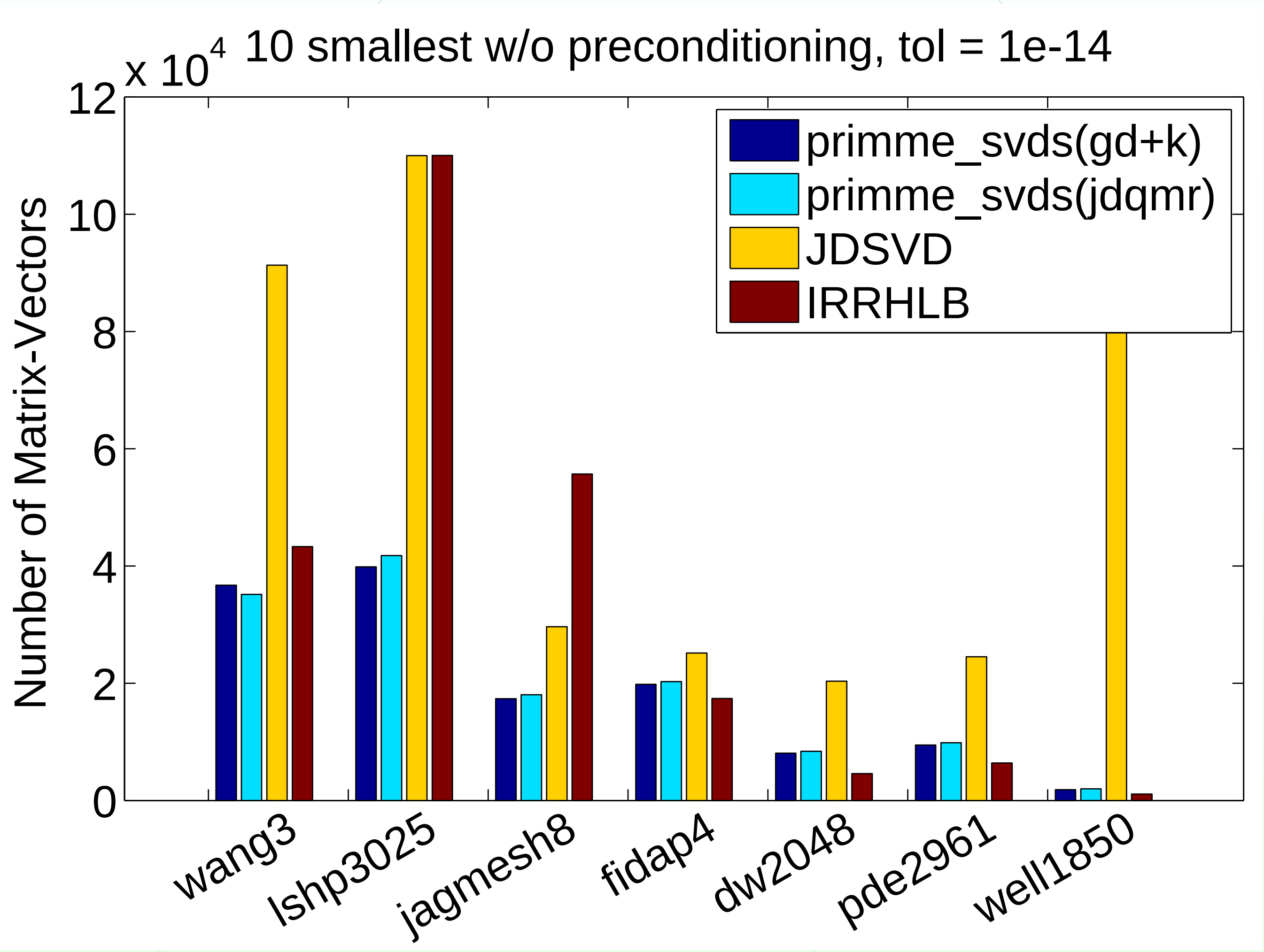
primme_svds (only first stage) is much faster in hard cases

Evaluation: Without preconditioning



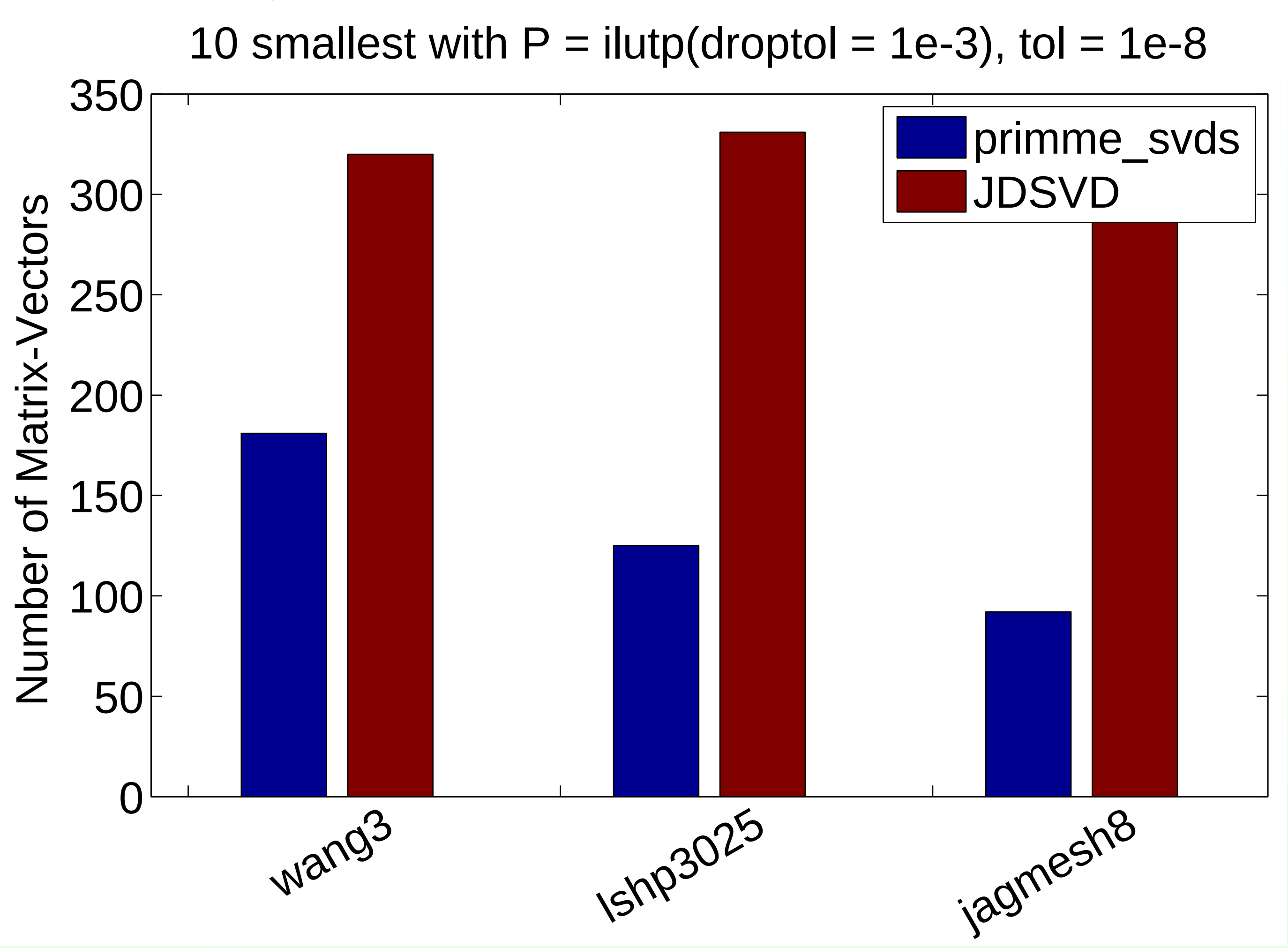
primme_svds (two stage) is superior for a few singular triplets

Evaluation: Without preconditioning



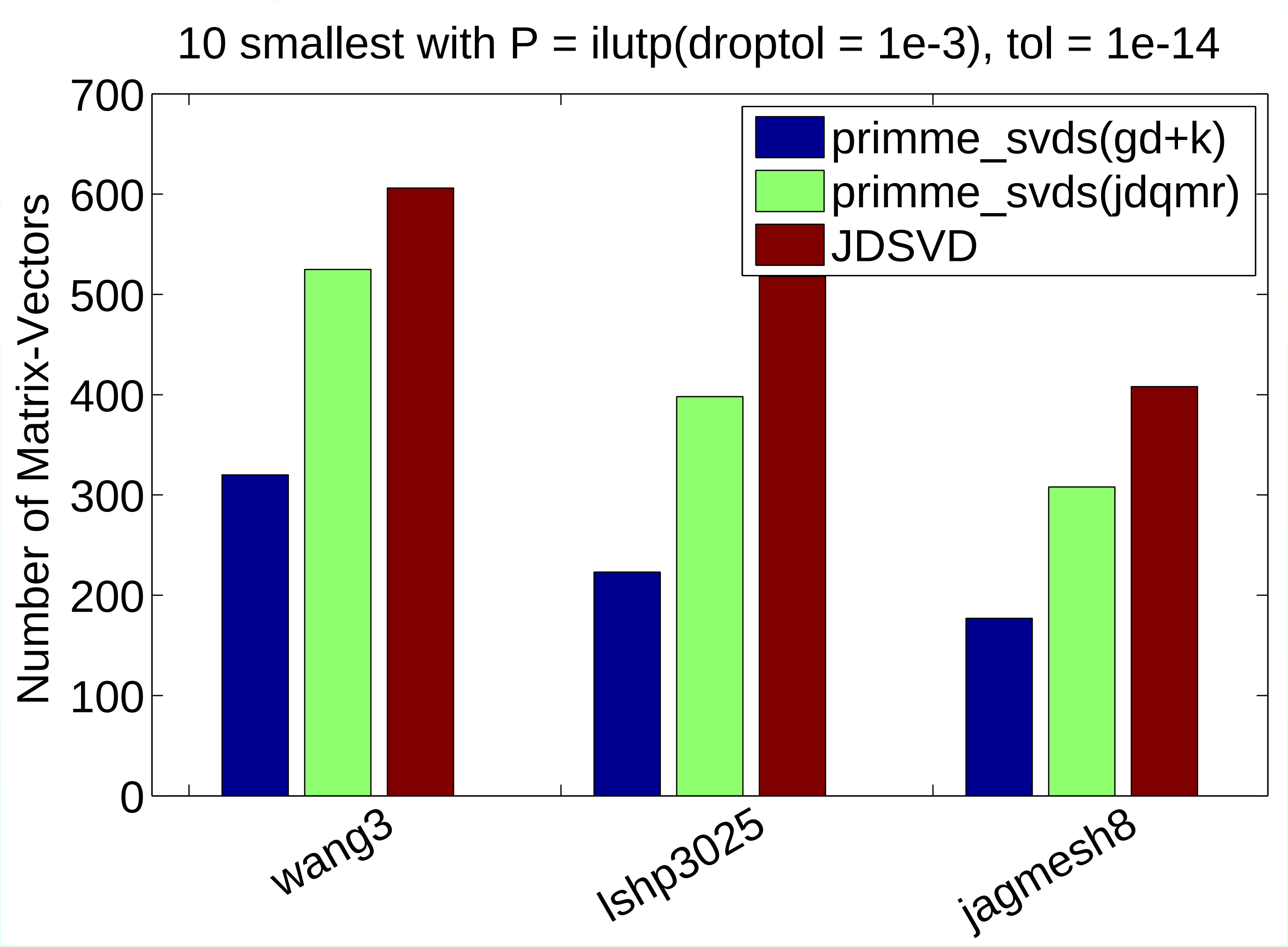
primme_svds (two stage) is much faster in hard cases

Evaluation: With preconditioning



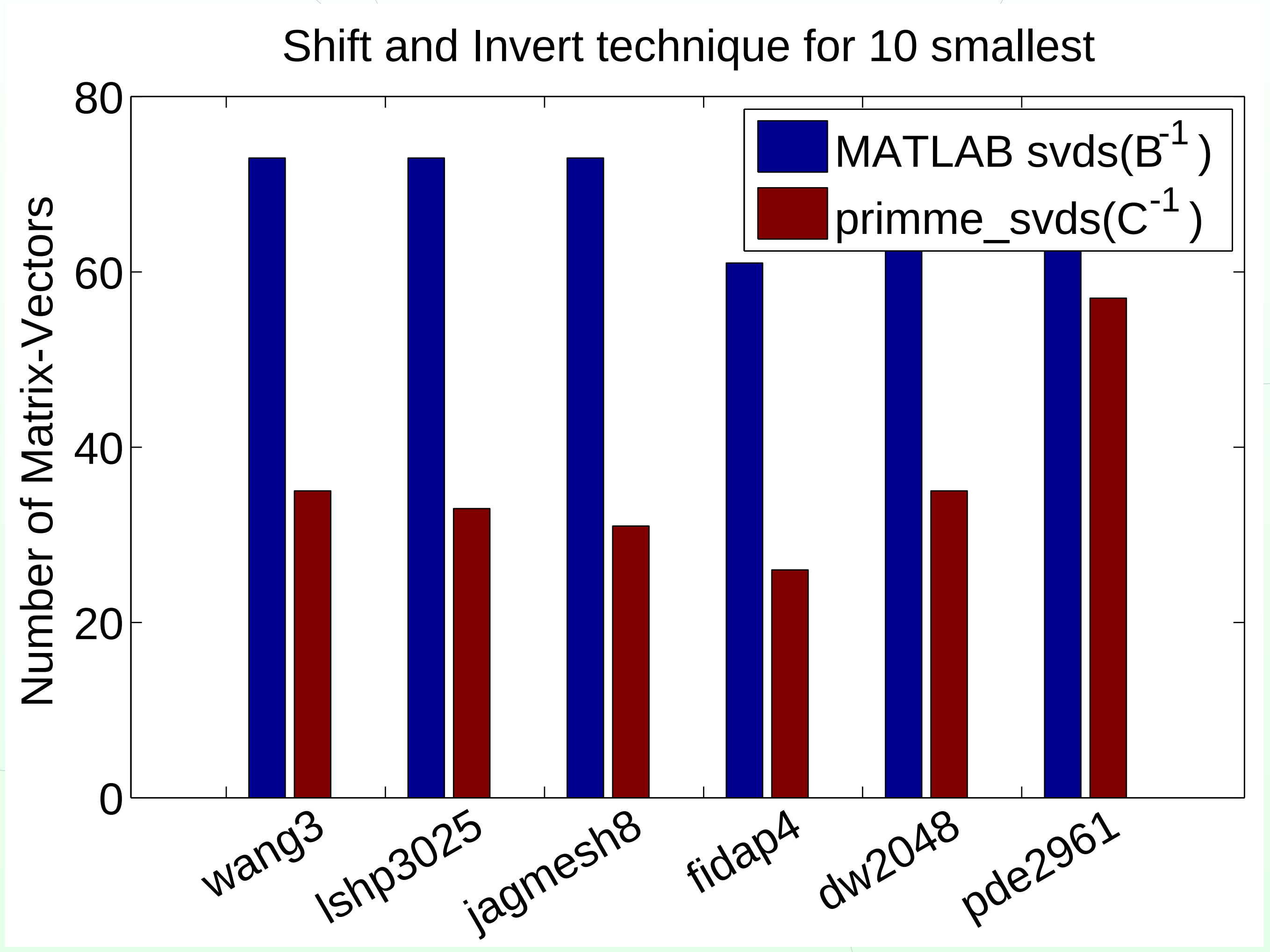
primme_svds (only first stage) is at least two times faster than JDSVD

Evaluation: With preconditioning



primme_svds (two stage) is faster than JDSVD

Evaluation: Shift and invert technique



primme_svds on C is faster than MATLAB svds

Conclusions

- primme_svds: a meta-method to compute a few singular triplets based on state-of-the-art eigensolver PRIMME

Conclusions

- primme_svds: a meta-method to compute a few singular triplets based on state-of-the-art eigensolver PRIMME
- Key idea: a two-stage strategy
  - take advantage of faster convergence on normal equations matrix

Conclusions

- primme_svds: a meta-method to compute a few singular triplets based on state-of-the-art eigensolver PRIMME
- Key idea: a two-stage strategy
  - take advantage of faster convergence on normal equations matrix
  - resolve remaining accuracy by exploiting power of PRIMME and refined projection on augmented matrix

Conclusions

- primme_svds: a meta-method to compute a few singular triplets based on state-of-the-art eigensolver PRIMME
- Key idea: a two-stage strategy
  - take advantage of faster convergence on normal equations matrix
  - resolve remaining accuracy by exploiting power of PRIMME and refined projection on augmented matrix
  - Any stage has flexibility to be replaced by other better methods

Conclusions

- primme_svds: a meta-method to compute a few singular triplets based on state-of-the-art eigensolver PRIMME
- Key idea: a two-stage strategy
  - take advantage of faster convergence on normal equations matrix
  - resolve remaining accuracy by exploiting power of PRIMME and refined projection on augmented matrix
  - Any stage has flexibility to be replaced by other better methods
- Shown efficiency and effectiveness both with and without preconditioning

Conclusions

- primme_svds: a meta-method to compute a few singular triplets based on state-of-the-art eigensolver PRIMME
- Key idea: a two-stage strategy
  - take advantage of faster convergence on normal equations matrix
  - resolve remaining accuracy by exploiting power of PRIMME and refined projection on augmented matrix
  - Any stage has flexibility to be replaced by other better methods
- Shown efficiency and effectiveness both with and without preconditioning
- A highly optimized production software enables the solution of large, real world problems

PRIMME

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

- **PRIMME including its MATLAB interface and primme_svds will be available this summer**
- **C implementation of primme_svds will be released with next version of PRIMME**

Download: `www.cs.wm.edu/~andreas/software`

Thank you for your attention!