A MATLAB interface for PRIMME for solving Eigenvalue and Singular Value problems

Andreas Stathopoulos and Lingfei Wu

Computer Science Department
College of William and Mary

Acknowledgment: National Science Foundation, DOE SciDAC
The problems

A large, sparse, Hermitian matrix:

Find \textit{nev} eigenvalues and corresponding eigenvectors

\[ Ax_i = \lambda_i x_i \]

A large, sparse, \( N \times M \) matrix

Find \textit{nev} singular values and corresponding left and right singular vectors

\[ Av_i = \sigma_i u_i \]

SVD problem solved as a Hermitian eigenvalue problem on

Normal equations \( A^T A \) or \( AA^T \),

Augmented matrix \([0 A; A^T 0]\)
Available software

Lanczos-based, no preconditioning

• ARPACK (Sorensen, Lehoucq)
• TRLAN (Wu, Simon)
• Industrial strength Lanczos (Grimes, Lewis, Simon)

Preconditioned eigensolver packages with various methods

• ANASAZI (Baker, Thornquist, Lehoucq, Hetmaniuk)
• PRIMME (AS, J.M.)
• SLEPc (Roman et al.)

Specific method preconditioned eigensolver software

• BLOPEX (Knyazev)
• JADAMILU (Bollhoefer, Notay)

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

- Near optimality through GD+k and JDQMR methods
- Over 12 methods accessible through PRIMME.
- Dynamic choice between best methods
- Block versions of methods
- Interior eigenvalues too
- Full set of defaults for non expert users
- Full customizability for expert users
- Parallel, high performance implementation
- C and Fortran interfaces, real and complex
- Accessible also in SLEPc

Download: www.cs.wm.edu/~andreas
PRIMME shown robust and efficient

Typically:

GD+1 smallest number of matrix-vector ops

JDMQR lowest time (if matrix sparse enough)
Dynamic method chooses between the fastest two algorithms

Matrix Fillet_13K: Dynamic matches best method

- GD+k
- JDQMR
- Dynamic
PRIMME multi-layer interface – End user

#include "primme.h"

primme_params primme;
primme_Initialize(&primme);

primme.n = n;
primme.numEvals = 20;

primme.matrixMatvec = MV(x,y,k)
primme.applyPreconditioner = PR(x,y,k)

ierr = dprimme(evals, evecs, rnorms, &primme);

Usually achieves full potential of the method
#include "primme.h"
primme_params primme;

primme.

outputFile = stdout  iseed = -1
printLevel = 5  restarting.scheme = primme_thick
numEvals = 10  restarting.maxPrevRetain = 1
aNorm = 1.0  correction.precondition = 1
eps = 1.0e-12  correction.robustShifts = 1
maxBasisSize = 15  correction.maxInnerIterations = -1
minRestartSize = 7  correction.relTolBase = 1.5
maxBlockSize = 1  correction.convTest = adaptive_ETolerance
maxOuterIterations = 10000
maxMatvecs = 300000  correction.projectors.LeftQ = 1
target = primme_smallest  correction.projectors.LeftX = 1
numTargetShifts = 0  correction.projectors.RightQ = 0
targetShifts = 1.0 2.0  correction.projectors.SkewQ = 0
locking = 1  correction.projectors.RightX = 1
initSize = 0  correction.projectors.SkewX = 1
numOrthoConst = 0;

matrixMatvec = MV(x,y,k)  applyPreconditioner = PR(x,y,k)
ierr = dprimme(evals, evecs, rnorms, &primme);
PRIMME in MATLAB

Benefits to PRIMME users

• Optimized Sparse, Block Matrix-Vector (MV) function
• Optimized libraries for Sparse matrix inversion and ILU preconditioning
• Optimized BLAS/LAPACK libraries
• Ease of use/development
• Easier to build and experiment on a SVD solver

Benefits to MATLAB users

• Availability of a preconditioned eigensolver
• Availability of a preconditioned singular value solver
• As robust and easy to use as eigs() but much faster
Interface similar to `eigs()`

\[
\begin{align*}
[\text{evals}] &= \\
[\text{evecs, evals}] &= \\
[\text{evecs, evals, resnorms}] &= \\
[\text{evecs, evals, resnorms, primmeStats}] &= \\
&\quad \text{primme_eigs}(A) \\
&\quad \text{primme_eigs}(A, \text{numEvals}) \\
&\quad \text{primme_eigs}(A, \text{numEvals, target}) \\
&\quad \text{primme_eigs}(A, \text{numEvals, target, opts}) \\
&\quad \text{primme_eigs}(A, \text{numEvals, target, opts, eigsMethod}) \\
&\quad \text{primme_eigs}(A, \text{numEvals, target, opts, eigsMethod, P}) \\
&\quad \text{primme_eigs}(A, \text{numEvals, target, opts, eigsMethod, P1,P2}) \\
&\quad \text{primme_eigs}(A, \text{numEvals, target, opts, eigsMethod, Pfun}) \\
&\quad \text{primme_eigs}(Afun, \text{dim},\ldots)
\end{align*}
\]

We allow `Afun` even for smallest magnitude eigenvalues
PRIMME vs PRIMME MEX vs \texttt{eigs()}

- Five smallest eigenvalues — \texttt{eigs(Afun)} to avoid inversion
- PRIMME compiled -O3 including \texttt{SparseMatVec,BLAS,LAPACK}
- PRIMME MEX uses MATLAB’s \texttt{SparseMatVec, BLAS, LAPACK}

**Time ratios over MEX JDqmr**

- MATLAB Library benefits
- PRIMME MEX faster than \texttt{eigs()}

![Graph showing time ratios over MEX JDqmr](image)
Performance of MATLAB’s Block Sparse Matvec

Block size of 2 biggest gain. Larger block no much better.
PRIMME MEX with block size of 2

In Block Krylov methods # Matvecs increase almost linearly with block size.

Using the MATLAB routines, block=2 offers better robustness with only small additional expense.

![Graph showing the ratio of time for block sizes 2, 3, and 4 over non-blocking for the number of eigenvalues of 3dspectralwave2 and Andrews.]
Using PRIMME for SVD

- Allow access to full functionality of PRIMME
- Allow choice of normal equations (ATA) or augmented matrix $B$ (OAAO)
- Allow for various preconditioning techniques:
  - $P \approx (A^TA)^{-1}$ or $P \approx B^{-1} = \begin{bmatrix} 0 & A^{-1} \\ A^{-T} & 0 \end{bmatrix}$ directly
  - $P \approx A^{-1}$ explicitly or through ILU: $P = U^{-1}L^{-1} \approx A^{-1}$. Use as:
    $PP^T \approx (A^TA)^{-1}$ or as $\begin{bmatrix} 0 & P \\ P^T & 0 \end{bmatrix} \approx B^{-1}$
    (matrix products not formed)
  - $P_{\text{fun}}$ user provided function for implementing any of the above $P$
Using PRIMME for SVD

\[ [S] = \]
\[ [U, S, V] = \]
\[ [U, S, V, norms, primeeout] = \]
primme_svds(A)
primme_svds(A, numSvs)
primme_svds(A, numSvs, target)
primme_svds(A, numSvs, target, opts)
primme_svds(A, numSvs, target, opts, eigsMethod)
primme_svds(A, numSvs, target, opts, eigsMethod, svdsMethod)
primme_svds(A, numSvs, target, opts, eigsMethod, svdsMethod, P)
primme_svds(A, numSvs, target, opts, eigsMethod, svdsMethod, P1,P2)
primme_svds(A, numSvs, target, opts, eigsMethod, svdsMethod, Pfun)
primme_svds(Afun, M, N,...)
What threshold to converge to?

Let \((u_i, \sigma_i, v_i)\) approximate singular triplet. Define:

\[
\begin{align*}
    r_v &= \|Av_i - \sigma_i u_i\| \\
    r_u &= \|A^T u_i - \sigma_i v_i\| \\
    r_{ATA} &= \|A^T Av_i - \sigma_i^2 v_i\| \\
    r_B &= \|B \begin{bmatrix} v \\ u \end{bmatrix} - \sigma_i \begin{bmatrix} v \\ u \end{bmatrix} \|
\end{align*}
\]

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

\[
r_u = \frac{r_{ATA}}{\sigma_i} = r_B \sqrt{2}
\]

Want to guarantee \(r_u < \delta \|A\|\), so converge each method to

\[
\begin{align*}
    r_{ATA} &< \sigma_i \delta \|A\| \\
    r_B &< \sqrt{2} \delta \|A\|
\end{align*}
\]
Which SVD method?

Convergence speed issue

ATA very fast for largest SVs (squared gaps)
slow for smallest but still much faster than OAAO

OAAO slower for largest eigenvalues
extremely slow and not robust for smallest (interior) SVs

Accuracy issue

OAAO can converge up to $\|A\| \epsilon_{mach}$

ATA can only converge up to $\|A\|^2 \epsilon_{mach}$
⇒ for small $\sigma_i$ cannot reach needed $\sigma_i \delta \|A\|$, if $\delta < \epsilon_{mach} \|A\|/\sigma_i$

Our PRIMME SVDS solution:

Use ATA to residual threshold max $(\sigma_i \delta \|A\|, \|A\|^2 \epsilon_{mach})$

Use OAAO to improve the ATA approximations to the required threshold
Accuracy limit and dynamic switching

\[ A = \text{diag}([1:10 \ 1000:100:1e6]) \], Prec = A + \text{rand}(1,10000) \ast 1e4 \\

\[ \|A^T u - \sigma v\| \\
\]

- OAAO
- ATA to limit
- ATA to \( \text{eps} \|A\|^2 \)
- switch to OAAO

Matvecs

\[ 0 \ 500 \ 1000 \ 1500 \ 2000 \ 2500 \ 3000 \]
Performance of primme\_svds

small matrices

When $A = LU$ or $A \approx LU$, store $L^T, U^T$ for faster memory access. Still less memory than svds inverting $B$. 
Performance of primme_svds medium size matrices — 1 smallest sv

Larger matrices where inversion possible -- nsvs = 1

Shift-invert speedup factor 4

When preconditioning effective speedup 1000.
Performance of `primme_svds` medium size matrices — 10 smallest svvs

Larger matrices where inversion possible -- nsvs = 10

Preconditioned benefits decrease when more values needed

Shift-invert still speedup 2-4
Performance of primme\_svds \hspace{1in} large size matrices — 4 smallest sv's

Factorization not possible, PRIMME the only alternative

<table>
<thead>
<tr>
<th>matrix:</th>
<th>cage14</th>
<th>dielFilterV2real</th>
<th>G3_circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>1,505,785</td>
<td>1,157,456</td>
<td>1,585,478</td>
</tr>
<tr>
<td>$\kappa(A)$</td>
<td>12</td>
<td>6.0E7</td>
<td>2.2E7</td>
</tr>
<tr>
<td>ilu(A)</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>primme_svds</td>
<td>128</td>
<td>3494</td>
<td>101595</td>
</tr>
<tr>
<td></td>
<td>61.2</td>
<td>4198</td>
<td>33077</td>
</tr>
</tbody>
</table>
Conclusions

- MATLAB interface to PRIMME
- MATLAB’s optimized libraries and sparse Matvec improve PRIMME
- PRIMME svds() more flexible and efficient than svds()
- Dynamic switching of ATA and OAAO methods in primme_svds()
Performance of *primme_svds*  
No preconditioning — 10 largest svvs

![Graph showing the ratio of time of `primme_svds()` to time of `eigs()` for different matrices.](image)
Performance of primme\_svds  

medium size matrices — 4 smallest sv$s$

---

Larger matrices where inversion possible —— nsvs = 4

<table>
<thead>
<tr>
<th></th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>cage11</td>
<td>100</td>
</tr>
<tr>
<td>finan512</td>
<td>1000</td>
</tr>
<tr>
<td>torso3</td>
<td>10000</td>
</tr>
<tr>
<td>wang3</td>
<td>100000</td>
</tr>
<tr>
<td>wang4</td>
<td>1000000</td>
</tr>
<tr>
<td>chipcool0</td>
<td>1000000</td>
</tr>
</tbody>
</table>

- SVDS($B^{-1}$)  
- primme\_svds($A^{-1}A^{-T}$)  
- primme\_svds(ILU precond)