Computing the SVD for large and sparse matrices

Rasmus Munk Larsen
SCCM & SOI-MDI
Stanford University

Lawrence Berkeley Lab, June 2000
Overview

- Introduction

- Golub-Kahan (Lanczos) bidiagonalization and semiorthogonalization

- PROPACK: Software for sparse SVD and eigenvalue problems

- Numerical experiments – comparisons between PROPACK, LANSO and ARPACK

- One-sided reorthogonalization

- Conclusion
The singular value decomposition (SVD)

Definition: Let \( A \) be a rectangular \( m \times n \) matrix with \( m \geq n \), then the SVD of \( A \) is

\[
A = U \Sigma V^T = \sum_{i=1}^{n} \sigma_i u_i v_i^T ,
\]

where the matrices \( U \in \mathbb{R}^{m \times m} \) and \( V \in \mathbb{R}^{n \times n} \) are orthogonal and

\[
\Sigma = \begin{bmatrix}
\Sigma_1 \\
0 
\end{bmatrix},
\]

where \( \Sigma_1 = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n) \) and

\[
\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0 ,
\]

\( r \) is the rank of \( A \).

The SVD has numerous applications in, e.g.,

- Information retrieval (LSI)
- Inverse problems (regularization)
- Statistics (PCA)
- Image and signal processing
Equivalent symmetric eigenvalue problems

Let the singular value decomposition of the $m \times n$ matrix $A$ be

$$A = U \Sigma V^T$$

and assume without loss of generality that $m \geq n$. Then

$$V^T (A^T A) V = \text{diag}(\sigma_1^2, \ldots, \sigma_n^2),$$

$$U^T (A A^T) U = \text{diag}(\sigma_1^2, \ldots, \sigma_n^2, 0, \ldots, 0).$$

Moreover, if $U = \begin{bmatrix} U_1 & U_2 \end{bmatrix}$ and

$$Y = \frac{1}{\sqrt{2}} \begin{bmatrix} U_1 & U_1 & \sqrt{2} U_2 \\ V & -V & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix},$$

then the orthonormal columns of the $(m + n) \times (m + n)$ matrix $Y$ form an eigenvector basis for the 2-cyclic matrix $C$ and

$$Y^T C Y = \text{diag}(\sigma_1, \ldots, \sigma_n, -\sigma_1, \ldots, -\sigma_n, 0, \ldots, 0).$$
SVD using sparse symmetric eigensolvers

Many sophisticated software packages exist for the symmetric eigenvalue problem. To mention a few:

- Lehoucq, Sorensen & Yang 1992 – 1997: (P)ARPACK
- Parlett, Simon, Wu et al. 1984 – 1999: (P)LANSO & TRLAN
- Marques 1998: LZPACK & BLZPACK

In several studies (Berry 1992 (SVDPACK), Eldén & Lundström 1996) LANSO and ARPACK have proved highly efficient for computing a few singular triplets of large and sparse or structured matrices.

However, using a symmetric eigensolver as a “black box” for SVD has certain disadvantages.
Using a symmetric eigensolver as a “black box”

Method 0: $A^T A$

- Severe loss of accuracy of small singular values if $A$ is ill-conditioned.
- Fast when $n \ll m$ since only Lanczos vectors of length $n$ need to be stored.

Method 1: $C = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$

- Lanczos vectors have length $m + n \Rightarrow$ Waste of memory and unnecessary work in reorthogonalization.
- Ritz values converge to pairs of $\pm \sigma_i \Rightarrow$ Twice as many iterations are needed.

To (almost) get the best of both worlds: Combine **Lanczos bidiagonalization** (LBD) with the efficient **semi-orthogonalization** schemes developed for the symmetric eigenvalue problem.
Algorithm Bidiag1 (Paige & Saunders)

1. Choose a starting vector $p_0 \in \mathbb{R}^m$, and let $\beta_1 = \|p_0\|$, $u_1 = p_0/\beta_1$ and $v_0 \equiv 0$

2. for $i = 1, 2, \ldots, k$ do
   
   $r_i = A^T u_i - \beta_i v_{i-1}$
   
   $\alpha_i = \|r_i\|
   
   $v_i = r_i/\alpha_i$
   
   $p_i = A v_i - \alpha_i u_i$
   
   $\beta_{i+1} = \|p_i\|$

   $u_{i+1} = p_i/\beta_{i+1}$

end

After $k$ steps we have the decomposition:

$$AV_k = U_{k+1} B_{k+1},$$

where $V_j$ and $U_{j+1}$ have orthonormal columns and

$$B_{k+1} = \begin{pmatrix}
\alpha_1 \\
\beta_2 \alpha_2 \\
\beta_3 \cdots \\
\vdots \\
\alpha_k \\
\beta_{k+1}
\end{pmatrix}$$
Semiorthogonality, fundamental result

Simon ’84: Let

\[ HQ_k = Q_k T_k + \beta_{k+1} q_{k+1} e_k^T \]

be the tridiagonal decomposition computed after \( k \) steps of the Lanczos algorithm on the hermitian matrix \( H \). If the columns of \( Q_k \) are kept semiorthogonal, i.e.,

\[
\max_{1 \leq i, j \leq k} |q_i^T q_j| \leq \sqrt{u/k} \quad \text{for } i \neq j ,
\]

then

\[
\tilde{Q}_k^T H \tilde{Q}_k = T_k + E_k ,
\]

where \( Q_k = \tilde{Q}_k \tilde{R}_k \) is the compact QR-decomposition of \( Q_k \) and the elements of \( E_k \) is of order \( O(u\|H\|) \).

It follows (Wiedlandt-Hoffman) that \( \lambda(T_k) \) are Ritz values for \( H \) within \( O(u\|H\|) \).
Semiorthogonality in LBD

Bidiag1 is equivalent to performing $2k + 1$ steps of symmetric Lanczos with matrix $C$ and starting vector $(u_1, 0)^T \in \mathbb{R}^{m+n}$, thus Simon’s result gives

**Corollary:** Define the levels of orthogonality in Bidiag1 by $\nu_{ij} \equiv v_i^T v_j$ and $\mu_{ij} \equiv u_i^T u_j$. If

$$\max_{1 \leq i, j \leq k} |\nu_{ij}| \leq \sqrt{u/(2k + 1)} \quad \text{for } i \neq j,$$

$$\max_{1 \leq i, j \leq k+1} |\mu_{ij}| \leq \sqrt{u/(2k + 1)} \quad \text{for } i \neq j,$$

then

$$\tilde{U}_{k+1}^T A \tilde{V}_k = B_{k+1} + O(u\|A\|),$$

where $U_{k+1} = \tilde{U}_{k+1} J_{k+1}$ and $V_k = \tilde{V}_k K_k$ are the compact QR-factorization of $U_{k+1}$ and $V_k$.

Therefore $\sigma(B_{k+1})$ are Ritz values for $A$ within $O(u\|A\|)$. 
The “ω-recurrences” for LBD

In finite precision arithmetic:

\[
\alpha_j v_j = A^T u_j - \beta_j v_{j-1} + f_j \\
\beta_{j+1} u_{j+1} = A v_j - \alpha_j u_j + g_j ,
\]

where \(f_j\) and \(g_j\) represent round-off errors.

It is simple to show that \(\mu_{j+1,i}\) and \(\nu_{ji}\) satisfy the coupled recurrences:

\[
\beta_{j+1} \mu_{j+1,i} = \alpha_i \nu_{ji} + \beta_i \nu_{j,i-1} - \alpha_j \mu_{ji} \\
+ u_i^T g_j - v_j^T f_i ,
\]

\[
\alpha_j \nu_{ji} = \beta_{i+1} \mu_{j,i+1} + \alpha_i \mu_{ji} - \beta_j \nu_{j-1,i} \\
- u_j^T g_i + v_i^T f_j ,
\]

where \(\mu_{ii} = \nu_{ii} = 1\) and \(\mu_{0i} = \nu_{0i} \equiv 0\) for \(1 \leq i \leq j\).

These recurrences were derived independently by Simon & Zha 1997.

**Partial reorthogonalization:** Use the recurrences to monitor the size of \(\mu_{j+1,i}\) and \(\nu_{ji}\). Reorthogonalize only when necessary.
Bounding the round-off terms

We can bound the size of the round-off term

\[ |u_i^T g_j - v_j^T f_i| \leq \|g_j\| + \|f_i\| \]
\[ \leq 4 \mathbf{u}((\alpha_j^2 + \beta_{j+1}^2)^{1/2} + (\alpha_i^2 + \beta_i^2)^{1/2}) + \epsilon_{MV} \]
\[ \equiv \tau \]

Round-off from matrix-vector multiply $\epsilon_{MV}$ is estimated conservatively: $\epsilon_{MV} \leq \mathbf{u}(\bar{n} + \bar{m}) \|A\|$, where $\bar{n}$ ($\bar{m}$) is the maximum number of non-zeros per row (column) in $A$.

Conservative updating rules $\nu_{j-1,i} \rightarrow \nu_{ji}$ and $\mu_{ji} \rightarrow \mu_{j+1,i}$:

\[
\begin{align*}
\nu'_{ji} & = \beta_{i+1}\mu_{j,i+1} + \alpha_i\mu_{ji} - \beta_j\nu_{j-1,i} \\
\nu_{ji} & = (\nu'_{ji} + \text{sign}(\nu'_{ji})\tau)/\alpha_j \\
\mu'_{j+1,i} & = \alpha_i\nu_{ji} + \beta_i\nu_{j,i-1} - \alpha_j\mu_{ji} \\
\mu_{j+1,i} & = (\mu'_{j+1,i} + \text{sign}(\mu'_{j+1,i})\tau)/\beta_{j+1}
\end{align*}
\]
Illustration of recurrences

Partial reorthogonalization (next slide) reduced the work compared to full reorthogonalization from 10100 → 926 inner products!
Outline of algorithm

*Lanczos bidiagonalization with Partial reorthogonalization:*

```latex
force = \text{FALSE}
\text{for } j = 1, 2, \ldots, k \text{ do}
\alpha_j v_j = A^T u_j - \beta_j v_{j-1}
\text{Update } \nu_{j-1,i} \rightarrow \nu_{ji}
\text{if } \max_{1 \leq i < j} |\nu_{ji}| > \text{tol or force}
\quad \text{Reorthogonalize } v_j
\quad \text{force} = (\max_{1 \leq i < j} |\nu_{ji}| > \text{tol})
\text{end}
\beta_{j+1} u_{j+1} = A v_j - \alpha_j u_j
\text{Update } \mu_{ji} \rightarrow \mu_{j+1,i}
\text{if } \max_{1 \leq i < j+1} |\mu_{j+1,i}| > \text{tol or force}
\quad \text{Reorthogonalize } u_{j+1}
\quad \text{force} = (\max_{1 \leq i < j+1} |\mu_{j+1,i}| > \text{tol})
\text{end}
\text{end}
```

- The variable “force” causes extra reorthogonalizations, which are necessary due to the coupling between $\nu_{ji}$ and $\mu_{j+1,i}$.
- **It is not correct simply to replace** “Reorthogonalize $u_{j+1}$” with “Reorthogonalize $u_j$ and $u_{j+1}$” and “Reorthogonalize $v_j$” with “Reorthogonalize $v_{j-1}$ and $v_j$”.

•
Estimated level of orthogonality...and the truth

\[ \log_{10} |v_i^j|, \text{ calculated using recurrence.} \]

\[ \log_{10} |v_i^j|, \text{ True v.} \]
Software

PROPACK: Software package written in Matlab.

Main components:

lanpro : Hermitian Lanczos with PRO
lanbpro : Lanczos bidiagonalization with PRO
lansvd : Singular value decomposition
laneig : Hermitian eigensolver ($\approx$LANSO)

Important implementation details:

— respecting coupling between $\mu$ and $\nu$
— extended local reorthogonalization
— iterated Gram-Schmidt reorth. (DGKS, BLAS-2)
— recovery from near zero $\alpha_i$ or $\beta_i$
— proper estimation of $\|A\|$
— interface similar to IRAM routines eigs and svds

URL: [http://soi.stanford.edu/~rmunk/PROPACK](http://soi.stanford.edu/~rmunk/PROPACK)

Fortran 77 versions of lanbpro and lansvd are also available upon request.
Numerical Experiments

The algorithms were tested by computing the first 10 singular triplets for the following real non-symmetric matrices from Matrix Market:

<table>
<thead>
<tr>
<th>Name</th>
<th>m</th>
<th>n</th>
<th>nnz(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WELL1850</td>
<td>1850</td>
<td>712</td>
<td>8758</td>
</tr>
<tr>
<td>ILLC1850</td>
<td>1850</td>
<td>712</td>
<td>8758</td>
</tr>
<tr>
<td>TOLS4000</td>
<td>4000</td>
<td>4000</td>
<td>8784</td>
</tr>
<tr>
<td>MHD4800A</td>
<td>4800</td>
<td>4800</td>
<td>102252</td>
</tr>
<tr>
<td>AF23560</td>
<td>23560</td>
<td>23560</td>
<td>460598</td>
</tr>
<tr>
<td>BCSSTK32</td>
<td>90449</td>
<td>90449</td>
<td>1921955</td>
</tr>
</tbody>
</table>

Software:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Matlab</th>
<th>Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lanczos bidiagonalization with PRO</td>
<td>lansvd</td>
<td>LANSVD</td>
</tr>
<tr>
<td>Lanczos with PRO on $A^T A$</td>
<td>laneig</td>
<td>LANSO</td>
</tr>
<tr>
<td>Lanczos with PRO on $C$</td>
<td>laneig</td>
<td>LANSO</td>
</tr>
<tr>
<td>IRAM on $A^T A$</td>
<td>eigs</td>
<td>ARPACK</td>
</tr>
<tr>
<td>IRAM on $C$</td>
<td>svds</td>
<td>ARPACK</td>
</tr>
</tbody>
</table>

Experimental setup: PC workstation with 600 MHz Pentium III CPU, 512MB memory, IEEE arithmetic, running RedHat Linux 6.2.
Performance of Matlab functions

The Matlab implementations were tested using Matlab 5.3 with LAPACK numerics library. The table shows execution time in seconds:

<table>
<thead>
<tr>
<th>Function</th>
<th>lansvd</th>
<th>laneig</th>
<th>eigs</th>
<th>svds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WELL1850</td>
<td>1.00</td>
<td>0.63</td>
<td>1.77</td>
<td>2.20</td>
</tr>
<tr>
<td>ILLC1850</td>
<td>0.57</td>
<td>0.37</td>
<td>1.75</td>
<td>1.39</td>
</tr>
<tr>
<td>TOLS4000</td>
<td>2.73</td>
<td>1.76</td>
<td>6.76</td>
<td>16.84</td>
</tr>
<tr>
<td>MHD4800A</td>
<td>1.52</td>
<td>1.41</td>
<td>3.31</td>
<td>4.38</td>
</tr>
<tr>
<td>AF23560</td>
<td>30.57</td>
<td>20.92</td>
<td>61.54</td>
<td>50.51</td>
</tr>
<tr>
<td>BCSSTK32</td>
<td>85.72</td>
<td>76.34</td>
<td>192.66</td>
<td>179.76</td>
</tr>
</tbody>
</table>

- \textit{laneig}(A^T A) wins on speed
- lansvd is a factor of 2 faster than laneig(C)
- PROPACK routines up to an order of magnitude faster than svds!
Performance, Fortran implementations

The Fortran version of lansvd was compared with the LANSO and ARPACK codes. The table shows execution (CPU) time in seconds:

<table>
<thead>
<tr>
<th>Program</th>
<th>LANSVD</th>
<th>LANSO</th>
<th>ARPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>$A$</td>
<td>$A^T A$</td>
<td>$C$</td>
</tr>
<tr>
<td>WELL1850</td>
<td>0.30</td>
<td>0.12</td>
<td>1.00</td>
</tr>
<tr>
<td>ILLC1850</td>
<td>0.19</td>
<td>0.12</td>
<td>0.61</td>
</tr>
<tr>
<td>TOLS4000</td>
<td>1.04</td>
<td>0.99</td>
<td>6.32</td>
</tr>
<tr>
<td>MHD4800A</td>
<td>0.51</td>
<td>0.38</td>
<td>1.22</td>
</tr>
<tr>
<td>AF23560</td>
<td>11.11</td>
<td>4.49</td>
<td>15.42</td>
</tr>
<tr>
<td>BCSSTK32</td>
<td>28.22</td>
<td>29.09</td>
<td>94.77</td>
</tr>
</tbody>
</table>

- LANSO($A^T A$) wins on speed
- LANSO consistently faster than ARPACK on same problem
- LANSVD significantly faster than any other backwards stable variants.

Experimental setup: EGCS (GNU) 2.95.2 compiler suite, ASCI Red BLAS by Greg Henry, LAPACK 3.0 compiled locally.
Tuning LANSO

The results for LANSO were improved by changing the strategy for expanding Lanczos basis.

Before:

```fortran
IF (NEIG.EQ.0) THEN
   LAST = FIRST+8
ELSE
   LAST = FIRST+MAX(2,((J-6)*(MAXPRS-NEIG))/NEIG)
ENDIF
```

After:

```fortran
IF (NEIG.EQ.0) THEN
   LAST = FIRST+MAX(2,FIRST/2)
ELSE
   LAST = FIRST+MAX(2,((J-6)*(MAXPRS-NEIG))/(2*NEIG+1))
ENDIF
```

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A^TA$</td>
<td>$C$</td>
</tr>
<tr>
<td>WELL1850</td>
<td>0.62</td>
<td>0.65</td>
</tr>
<tr>
<td>ILLC1850</td>
<td>1.28</td>
<td>0.73</td>
</tr>
<tr>
<td>TOLS4000</td>
<td>0.28</td>
<td>1.52</td>
</tr>
<tr>
<td>MHD4800A</td>
<td>0.46</td>
<td>1.45</td>
</tr>
<tr>
<td>AF23560</td>
<td>20.85</td>
<td>41.16</td>
</tr>
<tr>
<td>BCSSTK32</td>
<td>53.64</td>
<td>93.02</td>
</tr>
</tbody>
</table>
One-sided reorthogonalization

- Simon & Zha (1997): It is sufficient to keep either $U_{k+1}$ or $V_k$ orthogonal to compute accurate low rank approximations.

$$A \approx U_{k+1} B_k V_k^T$$

- It is also sufficient to compute accurate singular values of $A$.
- Leads to efficient algorithms for “skinny” matrices ($m \gg n$) $\Rightarrow$ only the short vectors $v_i$ need to be reorthogonalized.
- Only the current long vector $u_i$ need not be stored $\Rightarrow$ Storage requirements are low.
One-sided reorthogonalization

Singular values are just as accurate as with full or partial reorthogonalization. Example matrix not skinny enough to beat partial reorthogonalization in term of flops.
Conclusion

Experiments with matrices from different applications show that:

- PROPACK provides efficient and robust replacements for eigs and svds
- LANSVD is almost as fast as LANSO($A^T A$) (ratio $(m + n)/(2n)$)
- LANSVD is 2-4 times as fast LANSO($C'$) with same higher accuracy
- LANSVD and LANSO generally outperform ARPACK. Up to a factor of 2-3 on large examples.
  Caveats: No restarts = more memory!

Future work:

- Restarting (e.g. Thick Restarts as in TRLAN)
- Finish parallel implementation of LANSVD
- Add solver based on one-sided reorth. to PROPACK