The Jacobi method
Error and perturbation analysis

Zlatko Drmač

University of Zagreb
Department of Mathematics
visiting Virginia Tech

September 18. 2007
Outline

1. Jacobi method
2. One–sided Jacobi SVD
3. Floating point Jacobi
4. Provable accuracy
5. SVD computation in floating point
6. Scaling independent Jacobi SVD algorithm
Jacobi method

Ein leichtes Verfahren

One–sided Jacobi SVD

Floating point Jacobi

Provable accuracy

SVD computation in floating point

Scaling independent Jacobi SVD algorithm
Jacobi, 1844, 1846

Ein leichtes Verfahren ...

\[ H = H^T, \quad H^{(k+1)} = U_k^T H^{(k)} U_k \quad \Rightarrow \quad \Lambda = \text{diag}(\lambda_i) \quad (k \to \infty) \]

Each \( U_k \) annihilates \((p_k, q_k), (q_k, p_k)\) positions in \( H^{(k)} \).

\[
\cdots U_3^T U_2^T U_1^T \begin{pmatrix}
\bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet 
\end{pmatrix} U_1 U_2 U_3 \cdots = \begin{pmatrix}
\bullet & \ast & \times & 0 \\
\ast & \bullet & \ast & \ast \\
\times & \ast & \bullet & \ast \\
0 & \bullet & \bullet & \bullet 
\end{pmatrix}
\]

\[
U_1 = \begin{pmatrix}
\cos \psi_1 & \sin \psi_1 \\
-\sin \psi_1 & \cos \psi_1
\end{pmatrix} \bigoplus l_{n-2}, \quad U_2 = \cdots
\]

Jacobi rotation

\[
\cot 2\psi_k = \frac{H_{q_kq_k}^{(k)} - H_{p_kp_k}^{(k)}}{2 H_{p_kq_k}^{(k)}},
\]

\[
\tan \psi_k = \frac{\text{sign}(\cot 2\psi_k)}{|\cot 2\psi_k| + \sqrt{1 + \cot^2 2\psi_k}} \in \left(-\frac{\pi}{4}, \frac{\pi}{4}\right],
\]

\((p, q) = \mathcal{P}(k)\) pivot strategy, \( \mathcal{P} : \mathbb{N} \to \{(i, j) : i < j\} \)
Convergent strategies

Jacobi: \( |h_{pq}^{(k)}| = \max_{i \neq j} |h_{ij}^{(k)}|, \mathcal{P}(k) = (p, q). \)

Reading Jacobi’s 1846. paper recommended.

Cyclic: \( \mathcal{P} \) periodic, one full period called sweep.

Row–cyclic and column–cyclic:

\[
\begin{pmatrix}
\bullet & 1 \rightarrow & 2 \rightarrow & 3 \\
\bullet & \bullet & 4 \rightarrow & 5 \\
\bullet & \bullet & \bullet & 6 \\
\bullet & \bullet & \bullet & \bullet
\end{pmatrix},
\begin{pmatrix}
\bullet & 1 \downarrow & 2 \downarrow & 4 \downarrow \\
\bullet & \bullet & 3 \downarrow & 5 \downarrow \\
\bullet & \bullet & \bullet & 6 \downarrow \\
\bullet & \bullet & \bullet & \bullet
\end{pmatrix}
\]

\[
\text{Off}(H^{(k)}) = \sqrt{\sum_{i \neq j} (H^{(k)})_{ij}^2} \rightarrow 0 \ (k \rightarrow \infty)
\]

\( H^{(k)} \rightarrow \Lambda, U_1 \cdots U_k \cdots \rightarrow U \) as \( (k \rightarrow \infty) \); \( U^T H U = \Lambda \)

Asymptotically quadratic reduction of \( \text{Off}(H^{(k)}) \).

Forsythe, Henrici, Wilkinson, Rutishauser, Hari, Veselić

Asymptotically cubic strategies exist.
Jacobi method

One-sided Jacobi SVD

Floating point Jacobi

Provable accuracy

SVD computation in floating point

Scaling independent Jacobi SVD algorithm
**One–sided Jacobi SVD**

Hestenes used implicit Jacobi for SVD of $A \in \mathbb{R}^{m \times n}$:

Diagonalize $H = H^{(0)} = A^T A$; $A \equiv A_0$.

$H^{(1)} = V_0^T H^{(0)} V_0 = V_0^T A^T (AV_0) = A_1^T A_1$

$H^{(k+1)} = V_k^T H^{(k)} V_k = A_{k+1}^T A_{k+1} \longrightarrow \Lambda = \text{diag}(\lambda_i)$

↔ $A_{k+1} = A_k V_k$, where $H^{(k)} = A_k^T A_k$

$V_k$ uses Jacobi rotation to diagonalize

$$
\begin{pmatrix}
  h_{pp}^{(k)} & h_{pq}^{(k)} \\
  h_{qp}^{(k)} & h_{qq}^{(k)}
\end{pmatrix}
\quad h_{pp}^{(k)} = \| A_k(1 : m, p) \|^2
\quad h_{pq}^{(k)} = \| A_k(1 : m, q) \|^2
\quad h_{qp}^{(k)} = A_k(1 : m, p)^T A_k(1 : m, q)
$$

$h_{pp}^{(k)}$, $h_{qq}^{(k)}$ scalar update; $h_{pq}^{(k)}$ BLAS1 SDOT

$A_k \longrightarrow U \Sigma$, $\Sigma = \text{diag}(\sqrt{\lambda_i})$, $U^T U = I$

$V_1 \cdots V_k \cdots \longrightarrow V$, $V^T V = I$, $AV = U \Sigma$

$A = U \Sigma V^T$ the SVD of $A$. 

One–sided Jacobi SVD

Hestenes used implicit Jacobi for SVD of \( A \in \mathbb{R}^{m \times n} \):

Diagonalize \( H = H^{(0)} = A^T A; A \equiv A_0 \).

\[
H^{(1)} = V_0^T H^{(0)} V_0 = V_0^T A^T (AV_0) = A_1^T A_1
\]

\[
H^{(k+1)} = V_k^T H^{(k)} V_k = A_{k+1}^T A_{k+1} \longrightarrow \Lambda = \text{diag}(\lambda_i)
\]

\[
A_{k+1} = A_k V_k,
\]

where \( H^{(k)} = A_k^T A_k \)

\( V_k \) uses Jacobi rotation to diagonalize

\[
\begin{pmatrix}
  h_{pp}^{(k)} & h_{pq}^{(k)} \\
  h_{qp}^{(k)} & h_{qq}^{(k)}
\end{pmatrix}
\]

\[
h_{pp}^{(k)} = \| A_k(1 : m, p) \|^2
\]

\[
h_{qq}^{(k)} = \| A_k(1 : m, q) \|^2
\]

\[
h_{pq}^{(k)} = A_k(1 : m, p)^T A_k(1 : m, q)
\]

\( h_{pp}^{(k)}, h_{qq}^{(k)} \) scalar update; \( h_{pq}^{(k)} \) BLAS1 SDOT

\( A_k \longrightarrow U \Sigma, \Sigma = \text{diag}(\sqrt{\lambda_i}), U^T U = I \)

\( V_1 \cdots V_k \cdots \longrightarrow V, V^T V = I, AV = U \Sigma \)

\( A = U \Sigma V^T \) the SVD of \( A \).
One–sided rotation

\[
d_p = \|A_k(1:m,p)\|^2, \quad d_q = \|A_k(1:m,q)\|^2, \\
\xi = A_k(1:m,p)^T A_k(1:m,q);
\]

\[
\text{ROTATE}(A_{1:m,p}, A_{1:m,q}, d_p, d_q, \xi, [V_{1:m,p}, V_{1:m,p}])
\]

1: \quad \vartheta = \frac{d_q - d_p}{2 \cdot \xi}; \quad t = \frac{\text{sign}(\vartheta)}{|\vartheta| + \sqrt{1 + \vartheta^2}}; \\
\quad c = \frac{1}{\sqrt{1 + t^2}}; \quad s = t \cdot c; \\

2: \quad \left( A_{1:m,p} A_{1:m,q} \right) = \left( A_{1:m,p} A_{1:m,q} \right) \left[ \begin{array}{cc} c & s \\ -s & c \end{array} \right]; \\

3: \quad d_p = d_p - t \cdot \xi; \quad d_q = d_q + t \cdot \xi; \\

4: \quad \text{if } V \text{ is wanted then}

5: \quad \left( V_{1:n,p} V_{1:n,q} \right) = \left( V_{1:n,p} V_{1:n,q} \right) \left[ \begin{array}{cc} c & s \\ -s & c \end{array} \right] \\

6: \quad \text{end if}

Can avoid squared norms. Can use fast rotations. Unit stride memory access. Vectorizable. Parallelizable.
Jacobi SVD

\[ \hat{p} = n(n - 1)/2 ; s = 0 ; \text{convergence} = \text{false} ; \]
if \ V \text{ is wanted} \text{ then initialize } V = I_n \text{ end if}
for \ i = 1 \text{ to } n \text{ do } d_i = A_{1:m,i}^T A_{1:m,i} \text{ end for;}
repeat
\[ s = s + 1 ; p = 0 ; \]
for \ i = 1 \text{ to } n - 1 \text{ do }
\begin{align*}
&\text{for } j = i + 1 \text{ to } n \text{ do } \\
&\quad \xi = A_{1:m,i}^T A_{1:m,j} ; \\
&\quad \text{if } |\xi| > m\varepsilon \sqrt{d_i d_j} \text{ then } \\
&\qquad \text{call } \text{ROTATE}(A_{1:m,i}, A_{1:m,j}, d_i, d_j, \xi, [V_{1:m,i}, V_{1:m,j}]) ; \\
&\quad \text{else } p = p + 1 \text{ end if }
\end{align*}
end for
if \ p = \hat{p} \text{ then } \text{convergence} = \text{true} ; \text{ go to } \text{end if}
until \ s > 30
\begin{align*}
&\text{if } \text{convergence} \text{ then } \Sigma_{ii} = \sqrt{d_i} , \ U_{1:m,i} = A_{1:m,i} \Sigma_{ii}^{-1} , i = 1 : n ; \\
&\text{else Error: Did not converge in 30 sweeps. end if}
\end{align*}
Jacobi method

One–sided Jacobi SVD

Floating point Jacobi

Provable accuracy

SVD computation in floating point

Scaling independent Jacobi SVD algorithm
Jacobi in floating-point

Breakthrough: Jacobi method is more accurate than QR!

Demmel and Veselić: Let $\tilde{H}^{(k)}$, denote the computed matrices. Then, in the positive definite case, one step of Jacobi in floating-point arithmetic reads

$$\tilde{H}^{(k+1)} = \hat{U}_k^T (\tilde{H}^{(k)} + \delta \tilde{H}^{(k)}) \hat{U}_k$$

where $\hat{U}_k$ is exactly orthogonal and $\varepsilon$–close to the actually used Jacobi rotation $\tilde{U}_k$, and $\delta \tilde{H}^{(k)}$ is sparse with

$$e_k = \max_{i,j} \frac{|(\delta \tilde{H}^{(k)})_{ij}|}{\sqrt{(\tilde{H}^{(k)})_{ii}(\tilde{H}^{(k)})_{jj}}} \leq \epsilon$$

Relative perturbation of eigenvalues in the $k$–step bounded by $n e_k \| (\tilde{H}_s^{(k)})^{-1} \|_2$, $\tilde{H}_s^{(k)}$ scaled to have unit diagonal.

IMPORTANT: Stop when $\max_{i \neq j} |(\tilde{H}_s^{(k)})_{ij}| \leq \epsilon$

The accuracy depends on $\max_k \| (\tilde{H}_s^{(k)})^{-1} \|_2$
Jacobi in floating–point

If the entries of the initial $H$ are given with relative uncertainty $\varepsilon$, then:

- The spectrum is determined up to relative error of order of $n\varepsilon \|H_s^{-1}\|$ ($H_s$ diagonally scaled $H$ to have unit diagonal)
- The symmetric Jacobi method introduces perturbation of the order of $ne_k \max_k \|(\tilde{H}_s^{(k)})^{-1}\|_2$

Numerical evidence: $\max_k \|(\tilde{H}_s^{(k)})^{-1}\|_2$ behaves well.

Theoretical (still open) problem: Bound

$$\max_{k \geq 1} \frac{\|(H_s^{(k)})^{-1}\|_2}{\|H_s^{-1}\|_2} \quad \text{or} \quad \max_{k \geq 1} \frac{\kappa_2(H_s^{(k)})}{\kappa_2(H_s)}$$

Demmel, Veselić, Slapničar, Mascarenhas, Drmač
Jacobi method

One–sided Jacobi SVD

Floating point Jacobi

Provable accuracy

SVD computation in floating point

Scaling independent Jacobi SVD algorithm
Let $H = LL^T \succ 0$, $L$ Cholesky factor.

Use Veselić–Hari trick:

- If we apply Jacobi SVD to $L$, $LV = U\Sigma$, where $V$ is the product of Jacobi rotations, then $H = U\Sigma^2 U^T$.
- So, can apply Jacobi and get eigenvectors without accumulation of Jacobi rotations! This reduces flop count, memory requirements and memory traffic!
- This implicitly diagonalizes $L^T L$, which is similar to $H = LL^T$, and it is actually one step of the Rutishauser’s LR method. If $L$ is computed with pivoting, then $L^T L$ is ’more diagonal’ than $H$.
- The cost of Cholesky ($n^3 / 3$) much less than one sweep of Jacobi ($2n^3$ with fast rotations).
- In floating point

$$\tilde{L} \tilde{L}^T = H + \delta H, \quad \max_{i,j} \frac{|\delta H_{ij}|}{\sqrt{H_{ii}H_{jj}}} \leq \eta_C \lesssim n\epsilon$$
Provable accuracy

Now to the SVD of $\tilde{L}$:
One sided Jacobi SVD $\tilde{L} V_1 V_2 \cdots V_k \cdots V_\ell \rightarrow \tilde{U} \tilde{\Sigma}$

In floating point

- $\tilde{L} \leftarrow (((\tilde{L}_1 + \delta \tilde{L}_1) \hat{V}_1 + \delta \tilde{L}_2) \hat{V}_2 + \delta \tilde{L}_3) \hat{V}_3 + \cdots$

- If $y = x V$, $V$ rotation, $x$, $y$ row vectors, then $\tilde{y} = (x + \delta x) \hat{V}$, $\hat{V}$ orthogonal, $\|\delta x\| \leq 6 \varepsilon \|x\|$.

- Hence, each row of $\delta \tilde{L}_i$ is $\varepsilon$ small relative to the corresponding row of $\tilde{L}_i$. The $\hat{V}_j$ with $j \neq i$ do not change the row norms of $\delta \tilde{L}_i$.

- At convergence, $\tilde{U} \tilde{\Sigma} = (\tilde{L} + \delta \tilde{L}) \hat{V}$, with $\tilde{\Sigma} = \text{diag}(\tilde{\sigma}_i)$, $\|\delta \tilde{L}(i,:)\| \leq O(n)\varepsilon \|\tilde{L}(i,:)\|$ for all $i$.

- $\tilde{\lambda}_i = \tilde{\sigma}_i^2$ are the eigenvalues of $(\tilde{L} + \delta \tilde{L})(\tilde{L} + \delta \tilde{L})^T$
Provable accuracy

\[(\tilde{L} + \delta \tilde{L})(\tilde{L} + \delta \tilde{L})^T = \tilde{L}\tilde{L}^T + \tilde{L}\delta \tilde{L}^T + \delta \tilde{L}\tilde{L}^T + \delta \tilde{L} + \delta \tilde{L}^T\]

By Cauchy–Schwarz,

\[|E_{ij}| \leq 2O(n\varepsilon)\|\tilde{L}(i, :)\|\|\tilde{L}(j, :)\| + O(\varepsilon^2)\|\tilde{L}(i, :)\|\|\tilde{L}(j, :)\|\]

\[\approx (O(n\varepsilon) + O(\varepsilon^2))\sqrt{(\tilde{L}\tilde{L}^T)_{ii}(\tilde{L}\tilde{L}^T)_{jj}}\]

\[\approx (O(n\varepsilon) + O(\varepsilon^2))\sqrt{H_{ii}H_{jj}},\]

since \(\tilde{L}\tilde{L}^T = H + \delta H\), \(\max_{i,j} \frac{\|\delta H_{ij}\|}{\sqrt{H_{ii}H_{jj}}} \leq \eta C \approx n\varepsilon\)

So, we have the eigenvalues of

\[\tilde{L}\tilde{L}^T + E = H + \delta H + E = H + \Delta H, \quad \max_{i,j} \frac{\|\Delta H_{ij}\|}{\sqrt{H_{ii}H_{jj}}} \leq O(n\varepsilon)\]
Provable accuracy–conclusion

If $H \succ 0$ then

- The algorithm:
  1. Compute the Cholesky factorization $H = LL^T$;
  2. Compute $L = U\Sigma V^T$ using one–sided Jacobi SVD;
  3. Output: Set $\Lambda = \Sigma^2$; $H = U\Lambda U^T$

computes the eigenvalues and eigenvectors of $H$ with entry–wise small backward error $\max_{i,j} \frac{|\Delta H_{ij}|}{\sqrt{H_{ii}H_{jj}}} \leq O(n\varepsilon)$.

- The forward error is $\max_i |\delta \lambda_i|/\lambda_i \leq O(n^2\varepsilon)\|H_s^{-1}\|_2$.
- Most of the forward error comes from Step 1. Step 2. in floating point is as good as exact SVD.
- If Cholesky in Step 1 fails to compute $L$, then the matrix is entry–wise close to a non–definite matrix, and smallest eigenvalue can be lost due to symmetric tiny entry–wise perturbations.
- All computations in one $n \times n$ array.
Jacobi method

One–sided Jacobi SVD

Floating point Jacobi

Provable accuracy

SVD computation in floating point

Scaling independent Jacobi SVD algorithm
Let \( \text{rank}(A) = n \leq m \), \( D = \text{diag}(\|A(:, i)\|) \), and

\[
A \mapsto A + \delta A \quad \Rightarrow \quad \sigma_j \mapsto \sigma_j + \delta \sigma_j.
\]

\[
A + \delta A = (I + \delta AA^\dagger)A \Rightarrow \max_j \frac{|\tilde{\sigma}_j - \sigma_j|}{\sigma_j} \leq \|\delta AA^\dagger\|,
\]

\[
\|\delta AA^\dagger\| \leq \begin{cases} 
\frac{\|\delta A\|}{\|A\|}(\|A^\dagger\|\|A\|) = \epsilon \cdot \kappa(A), \\
\|\delta AD^{-1}\|\|(AD^{-1})^\dagger\|.
\end{cases}
\]

\[
\|\delta AD^{-1}\| \leq \sqrt{n} \max_j \frac{\|\delta A(:, j)\|}{\|A(:, j)\|} \leq \sqrt{n} \epsilon;
\]

\[
\|(AD^{-1})^\dagger\| \equiv \|A_s^\dagger\| \leq \sqrt{n} \min \Delta = \text{diag} \kappa(A\Delta)
\]

Possible: \( \|A_s^\dagger\| \ll \kappa(A) \); always \( \|A_s^\dagger\| \leq \sqrt{n} \kappa(A) \).

Jacobi SVD: \( \|A_s^\dagger\| \mapsto \) more accurate.

bidiagonal SVD: \( \kappa(A) \mapsto \) less accurate,

bidiagonalization provokes \( \kappa(A) \).

Jacobi++ SVD: \( A = D_1 CD_2 \rightarrow D_1(C + \delta C)D_2 \).
**Scalings: an example**

Fredholm integral equation of the first kind,

\[ y(\xi) = \int_a^b K(\xi, \zeta)x(\zeta) d\zeta \]

Here \( y \) denotes measured unknown function \( x \) distorted by the instrument with known kernel \( K(\cdot, \cdot) \). If the equation is discretized at \( \xi_1 < \cdots < \xi_m \), and the integral is computed using quadrature rule with the nodes \( \zeta_1 < \cdots < \zeta_n \) and weights \( d_1, \ldots, d_n \), then

\[ y(\xi_i) = \sum_{j=1}^{n} d_j K(\xi_i, \zeta_j)x(\zeta_j) + e_i, \quad e_i = \text{error}, \quad i = 1, \ldots, m. \]

Set \( y = (y(\xi_i))_{i=1}^{m} \), \( K = (K(\xi_i, \zeta_j)) \in \mathbb{R}^{m \times n} \). An approximation \( x = (x_j)_{j=1}^{n} \) of \( (x(\zeta_j))_{j=1}^{n} \) is obtained by ignoring the \( e_i \)'s and solving \( \|KDx - y\| \rightarrow \min \).

Thus, independence of column scaling means that the weights cannot spoil the solution of the algebraic problem.
Scalings: an example

This is important because the weights must cope e.g. with the problems of singular integrals and in this way we have complete freedom in choosing appropriate numerical integration formulae. Perturbation of $K$ is separated from $D$, $A + \delta A = (K + \delta K)D$.

Using sufficiently accurate quadrature, one obtains linear regression model $y = KDx + e$, $x \in \mathbb{R}^n$, $y, e \in \mathbb{R}^m$, with vector $e$ dominated by statistically independent measurement errors from $\mathcal{N}(0, S^2)$, where positive definite $S = \text{diag}(s_i)_{i=1}^n$ carries standard deviations of the $e_i$’s. A good estimate of $S$ is usually available. Wanted is an estimate $\tilde{x}$ of $x$. To normalize the error variances, the model is scaled with $S^{-1}$ to get $b = Ax + e'$, where $b = S^{-1}y$, $A = S^{-1}KD$, $e' = S^{-1}e$. 
Another example

\[
A = \begin{pmatrix}
1 & 1 & 1 \\
0 & 1 & \xi \\
0 & -1 & \xi
\end{pmatrix},
\text{where } \xi = 10/\text{eps}. \; \xi \approx 4.5e+016
\]

Givens rotation kills \( A_{13} \): \( \tilde{A}^{(1)} = \begin{pmatrix}
1 & \alpha & 0 \\
0 & \beta & \beta \\
0 & \beta & \beta
\end{pmatrix}; \)

\( \alpha \approx \sqrt{2}, \; \beta = 3.184525836262886e+016. \)

<table>
<thead>
<tr>
<th></th>
<th>( \text{svd}(A) )</th>
<th>( \text{svd}(A^T) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1 )</td>
<td>6.369051672525773e+16</td>
<td>6.369051672525772e+16</td>
</tr>
<tr>
<td>( \sigma_2 )</td>
<td>5.747279316501105e+00</td>
<td>3.004066501831585e+00</td>
</tr>
<tr>
<td>( \sigma_3 )</td>
<td>9.842664568695829e-01</td>
<td>4.220776043599739e-01</td>
</tr>
</tbody>
</table>

\[
\tilde{A}^{(1)} = \begin{pmatrix}
1 & \alpha & 0 \\
0 & \beta & \beta \\
0 & \beta & \beta
\end{pmatrix}, \quad \tilde{A}^{(2)} = \begin{pmatrix}
1 & \alpha & 0 \\
0 & \gamma & \gamma \\
0 & 0 & 0
\end{pmatrix},
\]

\( A = BD, \; \kappa_2(B) < 2. \)
A 2 × 2 example

Take in MATLAB

\[
A = \begin{pmatrix} 1.0e250 & 0 \\ 0 & 1.0e-201 \end{pmatrix},
\]

\(d = \text{diag}(A), \sigma = \text{svd}(A).\) \(A\) is (bi)diagonal, and its singular values are on the diagonal. However,

\[
d = \text{diag}(A) = \begin{pmatrix} 9.999999999999999e + 249 \\ 1.000000000000000e - 201 \end{pmatrix},
\]

\[
\sigma = \text{svd}(A) = \begin{pmatrix} 9.999999999999999e + 249 \\ 1.000000000000000e - 201 \end{pmatrix}.
\]

\[
\lambda = \text{eig}(A) = \begin{pmatrix} 9.999999999999999e + 249 \\ 1.000000000000000e - 201 \end{pmatrix}.
\]
The 2 × 2 example

LAPACK’s driver routine xGESVD computes $\alpha = \max_{i,j} |A_{ij}|$ and scales the input matrix $A$ with $(1/\alpha)\sqrt{\nu}/\varepsilon$ (if $\alpha < \sqrt{\nu}/\varepsilon$) or with $(1/\alpha)\varepsilon \sqrt{\omega}$ (if $\alpha > \varepsilon \sqrt{\omega}$). Here $\varepsilon$, $\nu$ and $\omega$ denote the round-off unit, underflow and overflow thresholds, respectively.

Let $\alpha = \max_{i,j} |A_{ij}|$, $\varepsilon = \text{eps}/2$, $\omega = \text{realmax}$, $\nu = \text{realmin}$, $s = \varepsilon \sqrt{\omega}/\alpha$, and scale $A$ with $s$. The singular values of $sA$ are on its diagonal; scaling the diagonal of $sA$ with $1/s$ changes the $(2,2)$ entry precisely to $1.000000000016167 \times 10^{-201}$. Five digits in the second singular value of a $2 \times 2$ diagonal matrix are lost due to scaling $\sigma = (1/s) \ast (s \ast d)$. (In MATLAB, $\omega \approx 1.79 \times 10^{308}$, $\nu \approx 2.22 \times 10^{-308}$.) The problem is not removed if $s$ is changed to the closest integer power of two.

Note that this scaling is designed to avoid overflow in the implicit use of $A^T A$. 
QRF preprocessor for Jacobi

\[ A = QR \ ; \ [\tilde{Q}, \tilde{R}] = qr(A), \tilde{Q}, \tilde{R} \text{ computed.} \]

Backward error analysis:

\[(\exists \delta A) \ (\exists \hat{Q}, \hat{Q}^T \hat{Q} = I) \ A + \delta A = \hat{Q} \tilde{R},\]

\[\| \delta A(:, i) \| \leq \epsilon_1 \| A(:, i) \|, \ i = 1, \ldots, n.\]

Perturbation analysis: \[\sigma_i(\tilde{R}) = \sigma_i((I + \delta AA^\dagger)A)\]

\[1 - \| \delta AA^\dagger \| \leq \frac{\sigma_i(\tilde{R})}{\sigma_i(A)} \leq 1 + \| \delta AA^\dagger \|, \ \text{for all } i.\]

Let \( A = A_s D, \ D = \text{diag}(\| A(:, i) \|). \)

\[\| \delta AA^\dagger \| = \| \delta AD^{-1}(AD^{-1})^\dagger \| \leq \sqrt{n} \max_i \frac{\| \delta A(:, i) \|}{\| A(:, i) \|} \| A_s^\dagger \|\]

\[\| A_s^\dagger \| \leq \kappa(A_s) \leq \sqrt{n} \min_{\Delta = \text{diag}} \kappa(A\Delta)\]

If \( \kappa(A_s) \) is moderate, then \( \text{SVD}(\tilde{R}) \) is OK for the \( \text{SVD}(A) \).
Strong backward stability

Jacobi SVD(\(\tilde{R}\)): \(\tilde{R}^T V = U\Sigma\). Computed:
[\(\tilde{U}, \tilde{V}, \tilde{\Sigma}\) = \text{JacobiSVD}(\(\tilde{R}^T\)).] Jacobi rotations \(\tilde{V}\) such that

\[
\max_{i \neq j} \left| \cos \angle((\tilde{U}\tilde{\Sigma})e_i, (\tilde{U}\tilde{\Sigma})e_j) \right| \leq O(n)u
\]

Error analysis:

\[
(\exists \delta \tilde{R}) \ (\exists \hat{V}, \hat{V}^T \hat{V} = I) \ (\tilde{R} + \delta \tilde{R})^T \hat{V} = (\tilde{U}\tilde{\Sigma})
\]

\[
\|\delta \tilde{R}(::, i)\| \leq \epsilon_2 \|\tilde{R}(::, i)\|, \ i = 1, \ldots, n.
\]

Finally,

\[
\tilde{R} + \delta \tilde{R} = \hat{Q}^T (A + \delta A) + \hat{Q}^T \hat{Q} \delta \tilde{R} = \hat{Q}^T (A + \underbrace{\delta A + \hat{Q} \delta \tilde{R}}_{\Delta A})
\]

where \(\|\Delta A(::, i)\| \leq (\epsilon_1 + \epsilon_2(1 + \epsilon_1))\|A(::, i)\|\) for all \(i\), and the SVD is \((A + \Delta A)^T \hat{Q} \hat{V} = \tilde{U}\tilde{\Sigma}\). Very nice and simple. Accurate.
The Fast drives out the Slow ...

... even if the Fast is Wrong. Wrong?
(From Kahan’s *The Baleful effect of computer benchmarks upon applied mathematics, physics and chemistry*)

Unfortunately, the Jacobi SVD algorithm is several times slower than QR, DC, DQD. As $m, n$ grow, Jacobi algorithm is 3, 5, 10 or more times slower. But, it is more accurate. But QR is faster. But Jacobi is more accurate. But DC is even faster. But Jacobi is ...

Strong mathematical advantage of the Jacobi algorithm (stability, accuracy) seems to be ignored because of its inefficiency in term of the run time, as compared to the QR, divide and conquer and other fast methods.

As a result, the Jacobi method is considered as method with no future, interesting for historical reasons and because of its simplicity.
How fassst is fassssssssst

- White (1958) describes the classical symmetric Jacobi on an IBM 650: *It requires 7 to 8 hours to complete the solution of a 26th order matrix.*
- Kogbetliantz (1955) on an IBM 701: *The type 701 performs with extreme rapidity: a 32 $\times$ 32 matrix is diagonalized in 19 minutes and the numerical results are printed in four minutes, the total time being 23 minutes.*
- Take random 1000 $\times$ 1000 $A = A_s D, \|A_s^+\| \approx 10^5, \kappa(D) \approx 10^6$: on a HP X2100, P4, 1Gb, MKL 5.1 (MKL 6.1) [GOTO] BLAS
time(Jacobi_SVD) = 33.11 (30.37) [35.03] sec.
time(SGESVD) = 11.86 (8.20) [6.80] sec.
time(SGESDD) = 6.95 (4.44) [2.85] sec.
accurate_digits(Jacobi_SVD) = 1234xxxx
accurate_digits(SGESVD, SDD) = xxxxxxxxx
How fasst is fassssssssst

- White (1958) describes the classical symmetric Jacobi on an IBM 650: *It requires 7 to 8 hours to complete the solution of a 26th order matrix*.
- Kogbetliantz (1955) on an IBM 701: *The type 701 performs with extreme rapidity*: a $32 \times 32$ matrix is diagonalized in 19 minutes and the numerical results are printed in four minutes, the total time being 23 minutes.
- Take random $1000 \times 1000 A = A_s D$, $\|A_s^\dagger\| \approx 10^5$, $\kappa(D) \approx 10^6$: on a HP X2100, P4, 1Gb, MKL 5.1 (MKL 6.1) [GOTO] BLAS
  
  \[
  \begin{array}{l}
  \text{time(Jacobi\_SVD)} = 33.11 \ (30.37) \ [35.03] \ \text{sec.} \\
  \text{time(SGESVD)} = 11.86 \ (8.20) \ [6.80] \ \text{sec.} \\
  \text{time(SGESDD)} = 6.95 \ (4.44) \ [2.85] \ \text{sec.} \\
  \text{accurate\_digits(Jacobi\_SVD)} = 1234xxxx \\
  \text{accurate\_digits(SGESVD, SDD)} = xxxxxxxxx
  \end{array}
  \]
How fassst is fassssssssst

- White (1958) describes the classical symmetric Jacobi on an IBM 650: *It requires 7 to 8 hours to complete the solution of a 26th order matrix.*
- Kogbetliantz (1955) on an IBM 701: *The type 701 performs with extreme rapidity: a $32 \times 32$ matrix is diagonalized in 19 minutes and the numerical results are printed in four minutes, the total time being 23 minutes.*
- Take random $1000 \times 1000$ $A = A_s D$, $\|A_s^\dagger\| \approx 10^5$, $\kappa(D) \approx 10^6$: on a HP X2100, P4, 1Gb, MKL 5.1 (MKL 6.1) [GOTO] BLAS
  
<table>
<thead>
<tr>
<th>Method</th>
<th>Time (sec)</th>
<th>Accurate Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi SVD</td>
<td>33.11 (30.37) [35.03]</td>
<td>1234xxxx</td>
</tr>
<tr>
<td>SGESVD</td>
<td>11.86 (8.20) [6.80]</td>
<td>xxxxxxxx</td>
</tr>
<tr>
<td>SGESDD</td>
<td>6.95 (4.44) [2.85]</td>
<td>xxxxxxxx</td>
</tr>
</tbody>
</table>
Accuracy: $\max_i \frac{\tilde{\sigma}_i - \sigma_i}{\sigma_i}$

Relative error: $A = BD$: Jacobi, SGESVD, SGESDD
Jacobi method
Zlatko Drmač

One-sided Jacobi SVD
Floating point
Provable accuracy
SVD computation in floating point
Scaling independent
Jacobi SVD algorithm

Improved efficiency

relative timings: SGEPVD vs. SGESVD (x) and SGESDD (+)

Timings: Jacobi/SGESVD, Jacobi/SGESDD
Why are bidiagonal SVD’s faster?

Bidiagonalization: $A = U_1 B V_1^t$ is

$$ A = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{bmatrix} \quad \rightarrow \quad B = \begin{bmatrix} \star & \star & 0 & 0 & 0 \\ 0 & \star & \star & 0 & 0 \\ 0 & 0 & \star & \star & 0 \\ 0 & 0 & 0 & \star & \star \\ 0 & 0 & 0 & 0 & \star \end{bmatrix} $$

- finite algorithm; $4mn^2 - 4n^3/3$ flops for $B$,
- 1.73 sec. for $1000 \times 1000$ (before 4.90, 2.79)
- optimized, BLAS 2.49, Howell, Demmel, Fulton, ..

SVD(B), $B = U_2 \Sigma V_2^t$ fast, mathematically sophisticated:
$O(n^3)$ in the QR SVD
$O(n^2)$ to $O(n^3)$ in the divide and conquer SVD
$O(n^2)$ Parlett and Dhillon MRRR; $O(n)$ data for $\Sigma$ only

Assemble: $U = U_1 U_2$, $V = V_1 V_2$. Use BLAS 3.
Why are bidiagonal SVD’s faster?

Bidiagonalization: \( A = U_1 B V_1^t \) is

\[
A = \begin{bmatrix}
  \bullet & \bullet & \bullet & \bullet & \bullet \\
  \bullet & \bullet & \bullet & \bullet & \bullet \\
  \bullet & \bullet & \bullet & \bullet & \bullet \\
  \bullet & \bullet & \bullet & \bullet & \bullet \\
  \bullet & \bullet & \bullet & \bullet & \bullet \\
\end{bmatrix} \quad \mapsto \quad B = \begin{bmatrix}
  \star & \star & 0 & 0 & 0 \\
  0 & \star & \star & 0 & 0 \\
  0 & 0 & \star & \star & 0 \\
  0 & 0 & 0 & \star & \star \\
  0 & 0 & 0 & 0 & \star \\
\end{bmatrix}
\]

- finite algorithm; \( 4mn^2 - 4n^3/3 \) flops for \( B \),
- 1.73 sec. for \( 1000 \times 1000 \) (before 4.90, 2.79)
- optimized, BLAS 2.49, Howell, Demmel, Fulton, ..

\( \text{SVD}(B) \), \( B = U_2 \Sigma V_2^t \) fast, mathematically sophisticated:
- \( O(n^3) \) in the QR SVD
- \( O(n^2) \) to \( O(n^3) \) in the divide and conquer SVD
- \( O(n^2) \) Parlett and Dhillon MRRR; \( O(n) \) data for \( \Sigma \) only

Assemble: \( U = U_1 U_2 \), \( V = V_1 V_2 \). Use BLAS 3.
... and the Jacobi SVD

\[ A^{(0)} = A ; \ V := I \]
\[ A^{(k+1)} = A^{(k)} V^{(k)} , \ V := V V^{(k)} , \ k = 0, 1, 2, \ldots \]

Under certain conditions, as \( k \to \infty \):

\[ A^{(k)} \to U \Sigma , \ (A^{(k)})^t A^{(k)} \to \Lambda = \Sigma^t \Sigma \]
\[ V^{(0)} V^{(1)} \ldots V^{(k)} \to V \]

- works with dense \( m \times n \) array \( A \) and dense \( n \times n \) array \( V \);
- busy memory traffic
- high flop count \( n(n - 1)/2 \) \( \times \) \((one \ dot \ product \ and \ two \ plane \ rotations) \approx 3mn^2 + 2n^3 \) flops per sweep. Many sweeps (> 5, 6, 10) needed for numerical convergence. \( mn^2 \) flops only to check convergence.
- all flops on BLAS 1 level
- destroys any initial zero structure

It does not seem to be fair to compare with the bidiagonal SVD’s in terms of efficiency. However, this Jacobi SVD can be considered naive.
What can be done?

- Precondition for better convergence (transform initial $A$ to better $X$, i.e. $X^T X$ structured and more diagonal than $A^T A$)
- Preprocess (make the $X$ above simpler, smaller, structured; exploit the structure)
- Reduce flop count (nontrivial modifications of the algorithm)
- Choose pivots wisely. Tailor pivot strategy for each given matrix. Go for higher order convergence. Avoid extra work. Stop just in time.
- Upgrade AMAP flops to BLAS $> 1$. Reduce memory traffic. Software engineering, profiling, ...

Z. Drmač and K. Veselić: a project with the Volkswagen Science Fundation.
A or $A^T$?

$$A = \begin{pmatrix} * & * & * \\ \bullet & * & * \\ \bullet & \bullet & * \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{pmatrix}, \quad A^T = \begin{pmatrix} * & \bullet & \bullet & \bullet & \bullet & \bullet \\ * & * & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{pmatrix}$$

If $m \gg n$, take $A$ and $A = QR$.
If $m \ll n$, take $A^T$ and $A^T = QR$.
If $m = n$ and $[A, A^T] \neq 0$ ($A^T A \neq AA^T$)... interesting. Subtle.
Let $H = A^T A$, $M = AA^T$. Which one is more diagonal? Which one is easier prey for the Jacobi algorithm? Ready to pay at most $O(n^2)$ for given $A$ ($O(n)$ for given $H$, $M$).

$$H = H^T = \begin{pmatrix} * & ? \\ ? & * \end{pmatrix}, \quad M = M^T = \begin{pmatrix} * & ? \\ ? & * \end{pmatrix}$$

$$\Omega(H) = \left\{ W^T HW : W^T W = I \right\}, \quad \text{adjoint orbit}$$

$M \in \Omega(H) \implies \|H\|_F = \|M\|_F$, thus

$$s(H) \equiv \sum_i h_{ii}^2 > \sum_i m_{ii}^2 \implies \sum_i h_{ii}^2 < \sum_{i \neq j} m_{ij}^2$$

$s(\cdot)$ attains maxima only at diagonal matrices in $\Omega(H)$. Interesting analysis (Brockett’s dynamical systems, gradient flow, critical points). Can be used to decide $A$ or $A^T$. Note that it estimates the off–norm, and not the scaled off–norm.
Let $H = U \Lambda U^T$ (spectral decomposition),

$$h_{ii} = \sum_{j=1}^{n} |u_{ij}|^2 \lambda_j, \quad i = 1, \ldots, n,$$

and

$$d(H) = (h_{11}, \ldots, h_{nn})^T, \quad \lambda(H) = (\lambda_1, \ldots, \lambda_n)^T.$$

$$d(H) = (U \circ U) \lambda(H) = S \lambda(H), \quad S = U \circ U.$$

$S$ is ortho–stochastic. Thus, $d(H)$ is majorised by $\lambda(H)$ ($d(H) \prec \lambda(H)$) (Schur theorem).

Normalize (2) using the trace of $H$,

$$\frac{d(H)}{\text{Trace}(H)} = S \frac{\lambda(H)}{\text{Trace}(H)},$$

then $d'(H)$ and $\lambda'(H)$ are two finite probability distributions connected via the doubly stochastic matrix $S$. 