Solution of eigenvalue problems

- Introduction – motivation
- Projection methods for eigenvalue problems
- Subspace iteration, The symmetric Lanczos algorithm
- Nonsymmetric Lanczos procedure;
- Implicit restarts
- Harmonic Ritz values, Jacobi-Davidson’s method
Background. Origins of Eigenvalue Problems

- Structural Engineering \[ Ku = \lambda Mu \] (Goal: frequency response)
- Electronic structure calculations [Schrödinger equation..]
- Stability analysis [e.g., electrical networks, mechanical system,..]
- Bifurcation analysis [e.g., in fluid flow]

- Large eigenvalue problems in quantum chemistry use up biggest portion of the time in supercomputer centers
Machine learning problems often require a (partial) Singular Value Decomposition.

Somewhat different issues in this case:

- Very large matrices, update the SVD
- Compute dominant singular values/vectors
- Many problems of approximating a matrix (or a tensor) by one of lower rank (Dimension reduction, ...)

But: Methods for computing SVD often based on those for standard eigenvalue problems
Background. The Problem(s)

- Standard eigenvalue problem:
  \[ Ax = \lambda x \]
  Often: \( A \) is symmetric real (or Hermitian complex)

- Generalized problem \( Ax = \lambda Bx \)
  Often: \( B \) is symmetric positive definite, \( A \) is symmetric or nonsymmetric

- Quadratic problems:
  \[(A + \lambda B + \lambda^2 C)u = 0\]

- Nonlinear eigenvalue problems (NEVP)
  \[
  \left[ A_0 + \lambda B_0 + \sum_{i=1}^{n} f_i(\lambda) A_i \right] u = 0
  \]
General form of NEVP

\[ A(\lambda)x = 0 \]

Nonlinear eigenvector problems:

\[
[A + \lambda B + F(u_1, u_2, \cdots, u_k)]u = 0
\]

What to compute:

- A few \( \lambda_i \)'s with smallest or largest real parts;
- All \( \lambda_i \)'s in a certain region of \( \mathbb{C} \);
- A few of the dominant eigenvalues;
- All \( \lambda_i \)'s (rare).
Some applications require the computation of a large number of eigenvalues and vectors of very large matrices.

Density Functional Theory in electronic structure calculations: ‘ground states’

Excited states involve transitions and invariably lead to much more complex computations. → Large matrices, *many* eigen-pairs to compute
**Background: The main tools**

**Projection process:**

(a) Build a ‘good’ subspace $K = \text{span}(V)$;

(b) get approximate eigenpairs by a Rayleigh-Ritz process:  
$\tilde{\lambda}, \tilde{u} \in K$ satisfy: 
$(A - \tilde{\lambda}I)\tilde{u} \perp K \rightarrow V^H(A - \tilde{\lambda}I)V y = 0$

$\tilde{\lambda} = \text{Ritz value}, \tilde{u} = V y = \text{Ritz vector}$

Two common choices for $K$:

1) Power subspace $K = \text{span}\{A^kX_0\}$; or $\text{span}\{P_k(A)X_0\}$;

2) Krylov subspace $K = \text{span}\{v, Av, \cdots, A^{k-1}v\}$
**Background. The main tools (cont)**

**Shift-and-invert:**

- If we want eigenvalues near $\sigma$, replace $A$ by $(A - \sigma I)^{-1}$.

**Example:** power method: $v_j = Av_{j-1}/\text{scaling}$ replaced by

$$v_j = (A - \sigma I)^{-1}v_{j-1} / \text{scaling}$$

- Works well for computing a few eigenvalues near $\sigma$.
- Used in commercial package NASTRAN (for decades!)
- Requires factoring $(A - \sigma I)$ (or $(A - \sigma B)$ in generalized case.) But convergence will be much faster.
- A solve each time - Factorization done once (ideally).
Deflation:

- Once eigenvectors converge remove them from the picture (e.g., with power method, second largest becomes largest eigenvalue after deflation).

Restarting Strategies:

- Restart projection process by using information gathered in previous steps

- ALL available methods use some combination of these ingredients.

[e.g. ARPACK: Arnoldi/Lanczos + ‘implicit restarts’ + shift-and-invert (option).]
Current state-of-the art in eigensolvers

- Eigenvalues at one end of the spectrum:
  - Subspace iteration + filtering [e.g. FEAST, Cheb,...]
  - Lanczos + variants (no restart, thick restart, implicit restart, Davidson,..), e.g., ARPACK code, PRIMME.
  - Block Algorithms [Block Lanczos, TraceMin, LOBPCG, SlepSc,..]
  - + Many others - more or less related to above

- ‘Interior’ eigenvalue problems (middle of spectrum):
  - Combine shift-and-invert + Lanczos/block Lanczos. Used in, e.g., NASTRAN
  - Rational filtering [FEAST, Sakurai et al.,... ]
**Projection Methods for Eigenvalue Problems**

**General formulation:**

Projection method onto $K$ orthogonal to $L$

- Given: Two subspaces $K$ and $L$ of same dimension.
- Find: $\tilde{\lambda}, \tilde{u}$ such that

$$\tilde{\lambda} \in \mathbb{C}, \tilde{u} \in K; \quad (\tilde{\lambda}I - A)\tilde{u} \perp L$$

**Two types of methods:**

Orthogonal projection methods: situation when $L = K$.

Oblique projection methods: When $L \neq K$. 
**Rayleigh-Ritz projection**

Given: a subspace $X$ known to contain good approximations to eigenvectors of $A$.

Question: How to extract good approximations to eigenvalues/eigenvectors from this subspace?

**Answer:** Rayleigh Ritz process.

Let $Q = [q_1, \ldots, q_m]$ an orthonormal basis of $X$. Then write an approximation in the form $\tilde{u} = Qy$ and obtain $y$ by writing

$$Q^H(A - \tilde{\lambda}I)\tilde{u} = 0$$

$$Q^H AQy = \tilde{\lambda}y$$
Procedure:
1. Obtain an orthonormal basis of $X$
2. Compute $C = Q^H A Q$ (an $m \times m$ matrix)
3. Obtain Schur factorization of $C$, $C = Y R Y^H$
4. Compute $\tilde{U} = Q Y$

Property: if $X$ is (exactly) invariant, then procedure will yield exact eigenvalues and eigenvectors.

Proof: Since $X$ is invariant, $(A - \tilde{\lambda} I) u = Q z$ for a certain $z$. $Q^H Q z = 0$ implies $z = 0$ and therefore $(A - \tilde{\lambda} I) u = 0$.

Can use this procedure in conjunction with the subspace obtained from subspace iteration algorithm
Subspace Iteration

- Original idea: projection technique onto a subspace if the form
  \[ Y = A^k X \]

- In practice: Replace \( A^k \) by suitable polynomial [Chebyshev]

Advantages:
- Easy to implement (in symmetric case);
- Easy to analyze;

Disadvantage: Slow.

- Often used with polynomial acceleration: \( A^k X \) replaced by 
  \( C_k(A)X \). Typically \( C_k = \) Chebyshev polynomial.
**Algorithm: Subspace Iteration with Projection**

1. **Start:** Choose an initial system of vectors $X = [x_0, \ldots, x_m]$ and an initial polynomial $C_k$.

2. **Iterate:** Until convergence do:
   
   (a) Compute $\hat{Z} = C_k(A)X_{old}$.
   
   (b) Orthonormalize $\hat{Z}$ into $Z$.
   
   (c) Compute $B = Z^H A Z$ and use the QR algorithm to compute the Schur vectors $Y = [y_1, \ldots, y_m]$ of $B$.
   
   (d) Compute $X_{new} = ZY$.
   
   (e) Test for convergence. If satisfied stop. Else select a new polynomial $C'_k$ and continue.
THEOREM: Let \( S_0 = \text{span}\{x_1, x_2, \ldots, x_m\} \) and assume that \( S_0 \) is such that the vectors \( \{Px_i\}_{i=1,\ldots,m} \) are linearly independent where \( P \) is the spectral projector associated with \( \lambda_1, \ldots, \lambda_m \). Let \( P_k \) be the orthogonal projector onto the subspace \( S_k = \text{span}\{X_k\} \). Then for each eigenvector \( u_i \) of \( A \), \( i = 1, \ldots, m \), there exists a unique vector \( s_i \) in the subspace \( S_0 \) such that \( Ps_i = u_i \). Moreover, the following inequality is satisfied

\[
\|(I - P_k)u_i\|_2 \leq \|u_i - s_i\|_2 \left(\left|\frac{\lambda_{m+1}}{\lambda_i}\right| + \epsilon_k\right)^k, \tag{1}
\]

where \( \epsilon_k \) tends to zero as \( k \) tends to infinity.
**Krylov subspace methods**

**Principle:** Projection methods on Krylov subspaces, i.e., on

\[ K_m(A, v_1) = \text{span}\{v_1, Av_1, \cdots, A^{m-1}v_1\} \]

- probably the most important class of projection methods [for linear systems and for eigenvalue problems]
- many variants exist depending on the subspace \( L \).

**Properties of** \( K_m \). Let \( \mu = \text{deg. of minimal polynom. of } v \). Then,

- \( K_m = \{p(A)v|p = \text{polynomial of degree } \leq m - 1\} \)
- \( K_m = K_\mu \) for all \( m \geq \mu \). Moreover, \( K_\mu \) is invariant under \( A \).
- \( \text{dim}(K_m) = m \) iff \( \mu \geq m \).
Arnoldi’s Algorithm

Goal: to compute an orthogonal basis of $K_m$.

Input: Initial vector $v_1$, with $\|v_1\|_2 = 1$ and $m$.

ALGORITHM : 1. Arnoldi’s procedure

For $j = 1, \ldots, m$ do

Compute $w := Av_j$

For $i = 1, \ldots, j$, do

$h_{i,j} := (w, v_i)$

$w := w - h_{i,j}v_i$

$h_{j+1,j} = \|w\|_2$; $v_{j+1} = w / h_{j+1,j}$

End
Result of Arnoldi’s algorithm

Let

\[
\overline{H}_m = \begin{bmatrix}
    x & x & x & x & x \\
    x & x & x & x & x \\
    x & x & x & x & x \\
    x & x & x & x & x \\
    x & x & x & x & x \\
\end{bmatrix} ; \quad H_m = \overline{H}_m(1 : m, 1 : m)
\]

1. \( V_m = [v_1, v_2, \ldots, v_m] \) orthonormal basis of \( K_m \).
2. \( AV_m = V_{m+1} \overline{H}_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T \)
3. \( V_m^T AV_m = H_m \equiv \overline{H}_m \) – last row.
Write approximate eigenvector as $\tilde{u} = V_m y$ + Galerkin condition

$$(A - \tilde{\lambda}I)V_m y \perp \mathcal{K}_m \rightarrow V_m^H (A - \tilde{\lambda}I)V_m y = 0$$

Approximate eigenvalues are eigenvalues of $H_m$

$$H_m y_j = \tilde{\lambda}_j y_j$$

Associated approximate eigenvectors are

$$\tilde{u}_j = V_m y_j$$

Typically a few of the outermost eigenvalues will converge first.


**Restarted Arnoldi**

In practice: Memory requirement of algorithm implies restarting is necessary

- Restarted Arnoldi for computing rightmost eigenpair:

  **ALGORITHM : 2. Restarted Arnoldi**

  1. **Start:** Choose an initial vector $v_1$ and a dimension $m$.
  2. **Iterate:** Perform $m$ steps of Arnoldi’s algorithm.
  3. **Restart:** Compute the approximate eigenvector $u_{1}^{(m)}$ associated with the rightmost eigenvalue $\lambda_{1}^{(m)}$.
  4. If satisfied stop, else set $v_1 \equiv u_{1}^{(m)}$ and goto 2.
**Example:**

Small Markov Chain matrix \([\text{Mark}(10), \text{dimension} = 55]\). Restarted Arnoldi procedure for computing the eigenvector associated with the eigenvalue with algebraically largest real part. We use \(m = 10\).

<table>
<thead>
<tr>
<th>(m)</th>
<th>(\Re(\lambda))</th>
<th>(\Im(\lambda))</th>
<th>Res. Norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.9987435899D+00</td>
<td>0.0</td>
<td>0.246D-01</td>
</tr>
<tr>
<td>20</td>
<td>0.9999523324D+00</td>
<td>0.0</td>
<td>0.144D-02</td>
</tr>
<tr>
<td>30</td>
<td>0.10000000368D+01</td>
<td>0.0</td>
<td>0.221D-04</td>
</tr>
<tr>
<td>40</td>
<td>0.10000000025D+01</td>
<td>0.0</td>
<td>0.508D-06</td>
</tr>
<tr>
<td>50</td>
<td>0.99999999996D+00</td>
<td>0.0</td>
<td>0.138D-07</td>
</tr>
</tbody>
</table>
Can be generalized to more than *one* eigenvector:

\[ \nu_1^{(\text{new})} = \sum_{i=1}^{p} \rho_i u_i^{(m)} \]

However: often does not work well – (hard to find good coefficients \( \rho_i \)'s)

Alternative: compute eigenvectors (actually Schur vectors) one at a time.

Implicit deflation.
Deflation

- Very useful in practice.
- Different forms: locking (subspace iteration), selective orthogonalization (Lanczos), Schur deflation, ...

A little background

Consider Schur canonical form

\[ A = URU^H \]

where \( U \) is a (complex) upper triangular matrix.

- Vector columns \( \mathbf{u}_1, \ldots, \mathbf{u}_n \) called Schur vectors.
- Note: Schur vectors are not unique. In particular, they depend on the order of the eigenvalues.
**Wiedlandt Deflation:** Assume we have computed a right eigenpair \(\lambda_1, u_1\). Wielandt deflation considers eigenvalues of

\[
A_1 = A - \sigma u_1 v^H
\]

Note:

\[
\Lambda(A_1) = \{\lambda_1 - \sigma, \lambda_2, \ldots, \lambda_n\}
\]

Wielandt deflation preserves \(u_1\) as an eigenvector as well all the left eigenvectors not associated with \(\lambda_1\).

- An interesting choice for \(v\) is to take simply \(v = u_1\). In this case Wielandt deflation preserves Schur vectors as well.
- Can apply above procedure successively.
ALGORITHM 3: Explicit Deflation

1. $A_0 = A$
2. For $j = 0 \ldots \mu - 1$ Do:
   3. Compute a dominant eigenvector of $A_j$
   4. Define $A_{j+1} = A_j - \sigma_j u_j u_j^H$
5. End

- Computed $u_1, u_2, \ldots$ form a set of Schur vectors for $A$.
- Alternative: implicit deflation (within a procedure such as Arnoldi).
When first eigenvector converges, put it in 1st column of $V_m = [v_1, v_2, \ldots, v_m]$. Arnoldi will now start at column 2, orthogonalizing still against $v_1, \ldots, v_j$ at step $j$.

Accumulate each new converged eigenvector in columns 2, 3, ... ['locked' set of eigenvectors.]

Thus, for $k = 2$:

$$V_m = \begin{bmatrix} v_1, v_2, \overbrace{v_3, \ldots, v_m}^{\text{active}} \\ \overbrace{v_1, v_2}^{\text{Locked}} \end{bmatrix}$$

$$H_m = \begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\
\end{pmatrix}$$
Similar techniques in Subspace iteration [G. Stewart’s SRRIT]

**Example:** Matrix Mark(10) – small Markov chain matrix ($N = 55$).

First eigenpair by iterative Arnoldi with $m = 10$.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$\Re(\lambda)$</th>
<th>$\Im(\lambda)$</th>
<th>Res. Norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.9987435899D+00</td>
<td>0.0</td>
<td>0.246D-01</td>
</tr>
<tr>
<td>20</td>
<td>0.9999523324D+00</td>
<td>0.0</td>
<td>0.144D-02</td>
</tr>
<tr>
<td>30</td>
<td>0.10000000368D+01</td>
<td>0.0</td>
<td>0.221D-04</td>
</tr>
<tr>
<td>40</td>
<td>0.1000000025D+01</td>
<td>0.0</td>
<td>0.508D-06</td>
</tr>
<tr>
<td>50</td>
<td>0.99999999996D+00</td>
<td>0.0</td>
<td>0.138D-07</td>
</tr>
</tbody>
</table>
Computing the next 2 eigenvalues of Mark(10).

<table>
<thead>
<tr>
<th>Eig.</th>
<th>Mat-Vec’s</th>
<th>( \text{Re}(\lambda) )</th>
<th>( \text{Im}(\lambda) )</th>
<th>Res. Norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>60</td>
<td>0.9370509474</td>
<td>0.0</td>
<td>0.870D-03</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td>0.9371549617</td>
<td>0.0</td>
<td>0.175D-04</td>
</tr>
<tr>
<td></td>
<td>78</td>
<td>0.9371501442</td>
<td>0.0</td>
<td>0.313D-06</td>
</tr>
<tr>
<td></td>
<td>87</td>
<td>0.9371501564</td>
<td>0.0</td>
<td>0.490D-08</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>0.8112247133</td>
<td>0.0</td>
<td>0.210D-02</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>0.8097553450</td>
<td>0.0</td>
<td>0.538D-03</td>
</tr>
<tr>
<td></td>
<td>112</td>
<td>0.8096419483</td>
<td>0.0</td>
<td>0.874D-04</td>
</tr>
<tr>
<td></td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td></td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td></td>
<td>152</td>
<td>0.8095717167</td>
<td>0.0</td>
<td>0.444D-07</td>
</tr>
</tbody>
</table>
Hermitian case: The Lanczos Algorithm

The Hessenberg matrix becomes tridiagonal:

\[ A = A^H \quad \text{and} \quad V_m^H A V_m = H_m \quad \rightarrow \quad H_m = H_m^H \]

We can write

\[
H_m = \begin{bmatrix}
\alpha_1 & \beta_2 \\
\beta_2 & \alpha_2 & \beta_3 \\
& \beta_3 & \alpha_3 & \beta_4 \\
& & \ddots & \ddots & \ddots \\
& & & \beta_m & \alpha_m
\end{bmatrix}
\]  

Consequence: three term recurrence

\[
\beta_{j+1}v_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}
\]
ALGORITHM 4. \textit{Lanczos}

1. Choose $v_1$ of norm unity. Set $\beta_1 \equiv 0, v_0 \equiv 0$
2. For $j = 1, 2, \ldots, m$ Do:
3. \hspace{1em} $w_j := Av_j - \beta_j v_{j-1}$
4. \hspace{1em} $\alpha_j := (w_j, v_j)$
5. \hspace{1em} $w_j := w_j - \alpha_j v_j$
6. \hspace{1em} $\beta_{j+1} := \|w_j\|_2$. If $\beta_{j+1} = 0$ then Stop
7. \hspace{1em} $v_{j+1} := w_j / \beta_{j+1}$
8. EndDo

\begin{center}
\textbf{Hermitian matrix + Arnoldi $\rightarrow$ Hermitian Lanczos}
\end{center}

- In theory $v_i$'s defined by 3-term recurrence are orthogonal.
- However: in practice severe loss of orthogonality;
Lanczos with reorthogonalization

Observation [Paige, 1981]: Loss of orthogonality starts suddenly, when the first eigenpair converges. It indicates loss of linear independence of the $v_i$'s. When orthogonality is lost, then several copies of the same eigenvalue start appearing.

- Full reorthogonalization – reorthogonalize $v_{j+1}$ against all previous $v_i$'s every time.
- Partial reorthogonalization – reorthogonalize $v_{j+1}$ against all previous $v_i$'s only when needed [Parlett & Simon]
- Selective reorthogonalization – reorthogonalize $v_{j+1}$ against computed eigenvectors [Parlett & Scott]
- No reorthogonalization – Do not reorthogonalize - but take measures to deal with 'spurious' eigenvalues. [Cullum & Willoughby]
Partial reorthogonalization

- Partial reorthogonalization: reorthogonalize only when deemed necessary.
- Main question is when?
- Uses an inexpensive recurrence relation
- Work done in the 80's [Parlett, Simon, and co-workers] + more recent work [Larsen, ’98]
- Package: PROPACK [Larsen] V 1: 2001, most recent: V 2.1 (Apr. 05)
- Often, need for reorthogonalization not too strong
The Lanczos Algorithm in the Hermitian Case

Assume eigenvalues sorted increasingly

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$$

- Orthogonal projection method onto $K_m$;
- To derive error bounds, use the Courant characterization

$$\tilde{\lambda}_1 = \min_{u \in K, u \neq 0} \frac{(Au, u)}{(u, u)} = \frac{(A\tilde{u}_1, \tilde{u}_1)}{(	ilde{u}_1, \tilde{u}_1)}$$

$$\tilde{\lambda}_j = \min_{\left\{ \begin{array}{l} u \in K, u \neq 0 \vspace{1pt} \\ u \perp \tilde{u}_1, \ldots, \tilde{u}_{j-1} \end{array} \right\}} \frac{(Au, u)}{(u, u)} = \frac{(A\tilde{u}_j, \tilde{u}_j)}{(	ilde{u}_j, \tilde{u}_j)}$$
Bounds for $\lambda_1$ easy to find – similar to linear systems.

Ritz values approximate eigenvalues of $A$ inside out:
Theorem [Kaniel, 1966]:

\[ 0 \leq \lambda^{(m)}_1 - \lambda_1 \leq (\lambda_N - \lambda_1) \left[ \frac{\tan \angle(v_1, u_1)}{T_{m-1}(1 + 2\gamma_1)} \right]^2 \]

where \( \gamma_1 = \frac{\lambda_2 - \lambda_1}{\lambda_N - \lambda_2} \); and \( \angle(v_1, u_1) \) = angle between \( v_1 \) and \( u_1 \).

+ results for other eigenvalues. [Kaniel, Paige, YS]

Theorem

\[ 0 \leq \lambda^{(m)}_i - \lambda_i \leq (\lambda_N - \lambda_1) \left[ \kappa_i^{(m)} \frac{\tan \angle(v_i, u_i)}{T_{m-i}(1 + 2\gamma_i)} \right]^2 \]

where \( \gamma_i = \frac{\lambda_{i+1} - \lambda_i}{\lambda_N - \lambda_{i+1}} \), \( \kappa_i^{(m)} = \prod_{j<i} \frac{\lambda_j^{(m)} - \lambda_N}{\lambda_j^{(m)} - \lambda_i} \)
The Lanczos biorthogonalization \((A^H \neq A)\)

ALGORITHM : 5. **Lanczos bi-orthogonalization**

1. Choose two vectors \(v_1, w_1\) such that \((v_1, w_1) = 1\).
2. Set \(\beta_1 = \delta_1 \equiv 0, w_0 = v_0 \equiv 0\)
3. For \(j = 1, 2, \ldots, m\) Do:
   4. \(\alpha_j = (Av_j, w_j)\)
   5. \(\hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}\)
   6. \(\hat{w}_{j+1} = A^T w_j - \alpha_j w_j - \delta_j w_{j-1}\)
   7. \(\delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}. \text{ If } \delta_{j+1} = 0 \text{ Stop}\)
   8. \(\beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}\)
   9. \(w_{j+1} = \hat{w}_{j+1}/\beta_{j+1}\)
10. \(v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}\)
11. EndDo
Builds a pair of biorthogonal bases for the two subspaces

\[ \mathcal{K}_m(A, v_1) \quad \text{and} \quad \mathcal{K}_m(A^H, w_1) \]

Many choices for \( \delta_{j+1}, \beta_{j+1} \) in lines 7 and 8. Only constraint:

\[ \delta_{j+1}\beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1}) \]

Let

\[
T_m = \begin{bmatrix}
\alpha_1 & \beta_2 \\
\delta_2 & \alpha_2 & \beta_3 \\
& \ddots & \ddots & \ddots \\
& & \ddots & \ddots & \ddots \\
& & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\
& & & & \delta_m & \alpha_m
\end{bmatrix}.
\]

\( v_i \in \mathcal{K}_m(A, v_1) \) and \( w_j \in \mathcal{K}_m(A^T, w_1) \).
If the algorithm does not break down before step $m$, then the vectors $v_i, i = 1, \ldots, m$, and $w_j, j = 1, \ldots, m$, are biorthogonal, i.e.,

$$(v_j, w_i) = \delta_{ij} \quad 1 \leq i, j \leq m .$$

Moreover, $\{v_i\}_{i=1,2,\ldots,m}$ is a basis of $\mathcal{K}_m(A, v_1)$ and $\{w_i\}_{i=1,2,\ldots,m}$ is a basis of $\mathcal{K}_m(A^H, w_1)$ and

\[
\begin{align*}
AV_m &= V_m T_m + \delta_{m+1} v_{m+1} e_m^H, \\
A^H W_m &= W_m T_m^H + \bar{\beta}_{m+1} w_{m+1} e_m^H, \\
W_m^H AV_m &= T_m .
\end{align*}
\]
If $\theta_j, y_j, z_j$ are, respectively an eigenvalue of $T_m$, with associated right and left eigenvectors $y_j$ and $z_j$ respectively, then corresponding approximations for $A$ are

<table>
<thead>
<tr>
<th>Ritz value</th>
<th>Right Ritz vector</th>
<th>Left Ritz vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_j$</td>
<td>$V_m y_j$</td>
<td>$W_m z_j$</td>
</tr>
</tbody>
</table>

[Note: terminology is abused slightly - Ritz values and vectors normally refer to Hermitian cases.]
Advantages and disadvantages

**Advantages:**
- Nice three-term recurrence – requires little storage in theory.
- Computes left and a right eigenvectors at the same time.

**Disadvantages:**
- Algorithm can break down or nearly break down.
- Convergence not too well understood. Erratic behavior.
- Not easy to take advantage of the tridiagonal form of $T_m$. 
Look-ahead Lanczos

Algorithm breaks down when:

\[(\hat{v}_{j+1}, \hat{w}_{j+1}) = 0\]

Three distinct situations.

- ‘lucky breakdown’ when either \(\hat{v}_{j+1}\) or \(\hat{w}_{j+1}\) is zero. In this case, eigenvalues of \(T_m\) are eigenvalues of \(A\).
- \((\hat{v}_{j+1}, \hat{w}_{j+1}) = 0\) but of \(\hat{v}_{j+1} \neq 0, \hat{w}_{j+1} \neq 0 \rightarrow \) serious breakdown. Often possible to bypass the step (+ a few more) and continue the algorithm. If this is not possible then we get an ...
- ... Incurable break-down. [very rare]
Look-ahead Lanczos algorithms deal with the second case. See Parlett 80, Freund and Nachtigal ’90.... Main idea: when break-down occurs, skip the computation of $v_{j+1}, w_{j+1}$ and define $v_{j+2}, w_{j+2}$ from $v_j, w_j$. For example by orthogonalizing $A^2v_j$ ... Can define $v_{j+1}$ somewhat arbitrarily as $v_{j+1} = Av_j$. Similarly for $w_{j+1}$.

- Drawbacks: (1) projected problem no longer tridiagonal (2) difficult to know what constitutes near-breakdown.