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
Origins of Eigenvalue Problems

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

Large sparse eigenvalue problems are among the most demanding calculations (in terms of CPU time) in scientific computing.
New application in information technology

- Search engines (google) rank web-sites in order to improve searches
- The google toolbar on some browsers (http://toolbar.google.com) - gives a measure of relevance of a page.
- The problem can be formulated as a Markov chain – Seek the dominant eigenvector
- Algorithm used: power method
- For details see: http://www.iprcom.com/papers/pagerank/index.html
We consider the eigenvalue problem

\[ Ax = \lambda x \text{ or } Ax = \lambda Bx \]

Typically: \( B \) is symmetric (semi) positive definite, \( A \) is symmetric or nonsymmetric

Requirements vary:

- Compute a few \( \lambda_i \)'s with smallest or largest real parts;
- Compute all \( \lambda_i \)'s in a certain region of \( \mathbb{C} \);
- Compute a few of the dominant eigenvalues;
- Compute all \( \lambda_i \)'s.
Types of problems

* Standard Hermitian (or symmetric real) $Ax = \lambda x$, $A^H = A$
* Standard non-Hermitian $Ax = \lambda x$, $A^H \neq A$
* Generalized

$$Ax = \lambda Bx$$

Several distinct sub-cases ($B$ SPD, $B$ SSPD, $B$ singular with large null space, both $A$ and $B$ singular, etc.)

* Quadratic

$$(A + \lambda B + \lambda^2 C)x = 0$$

* Nonlinear

$$A(\lambda)x = 0$$
General Tools for Solving Large Eigen-Problems

- Projection techniques – Arnoldi, Lanczos, Subspace Iteration;
- Preconditionings: shift-and-invert, Polynomials, ...
- Deflation and restarting techniques

Good computational codes combine these 3 ingredients
A few popular solution Methods

- Subspace Iteration [Now less popular – sometimes used for validation]
- Arnoldi’s method (or Lanczos) with polynomial acceleration [Stiefel ’58, Rutishauser ’62, YS ’84,’85, Sorensen ’89,...]
- Shift-and-invert and other preconditioners. [Use Arnoldi or Lanczos for \((A - \sigma I)^{-1}\).
- Davidson’s method and variants, Generalized Davidson’s method [Morgan and Scott, 89], Jacobi-Davidson
**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

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

**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

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

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

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

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

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

**Advantages:**

**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 $P_k$ 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

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

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 \( A \).
- \( \dim(K_m) = m \) 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

\[
\begin{aligned}
    h_{i,j} &:= (w, v_i) \\
    w &:= w - h_{i,j}v_i \\
    h_{j+1,j} &= \|w\|_2; \ v_{j+1} = w/h_{j+1,j}
\end{aligned}
\]

End
Result of Arnoldi’s algorithm

Let

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

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. \(V_m^T A V_m = H_m \equiv \overline{H}_m - \text{last row.}\)
Application to eigenvalue problems

- Write approximate eigenvector as $\tilde{u} = V_m y + \text{Galerkin condition}$

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

- 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>0.1000000025D+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>
Can be generalized to more than *one* eigenvector:

\[ v_1^{(new)} = \sum_{i=1}^{p} \rho_i u_i^{(m)} \]

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

\[ A = U R U^H \]

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

- Vector columns \( u_1, \ldots, u_n \) called Schur vectors.
- Note: Schur vectors depend on each other, and 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 $k = 2$:

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

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

**Example:** Matrix Mark(10) – small Markov chain matrix \( \mathbf{N} = 55 \).

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>0.9999999996D+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>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

\[ H_m = \begin{bmatrix}
\alpha_1 & \beta_2 \\
\beta_2 & \alpha_2 & \beta_3 \\
\beta_3 & \alpha_3 & \beta_4 \\
& \ddots & \ddots & \ddots \\
& & \beta_m & \alpha_m
\end{bmatrix} \]  \hspace{1cm} (2)

Consequence: three term recurrence

\[ \beta_{j+1} v_{j+1} = A v_j - \alpha_j v_j - \beta_j v_{j-1} \]
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. $w_j := Av_j - \beta_j v_{j-1}$

4. $\alpha_j := (w_j, v_j)$

5. $w_j := w_j - \alpha_j v_j$

6. $\beta_{j+1} := \|w_j\|_2$. If $\beta_{j+1} = 0$ then Stop

7. $v_{j+1} := w_j / \beta_{j+1}$

8. EndDo

Hermitian matrix + Arnoldi $\rightarrow$ Hermitian Lanczos

➢ 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

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

Ritz values approximate eigenvalues of $A$ inside out:
A-priori error bounds

Theorem [Kaniel, 1966]:

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

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

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

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. $\delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}$. If $\delta_{j+1} = 0$ 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

\[ T_m = \begin{bmatrix} \alpha_1 & \beta_2 \\ \delta_2 & \alpha_2 & \beta_3 \\ & \ddots & \ddots & \ddots \\ & & \ddots & \ddots & \ddots \end{bmatrix} \]

\[ \begin{bmatrix} \delta_{m-1} & \alpha_{m-1} & \beta_m \\ \delta_m & \alpha_m \end{bmatrix} \]

\( v_i \in \mathcal{K}_m(A, v_1) \) 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

$$AV_m = V_m T_m + \delta_{m+1} v_{m+1} e_m^H,$$
$$A^H W_m = W_m T_m^H + \bar{\beta}_{m+1} w_{m+1} e_m^H,$$
$$W_m^H AV_m = T_m.$$
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.
**Preconditioning eigenvalue problems**

- **Goal:** To extract good approximations to add to a subspace in a projection process. Result: faster convergence.

- **Best known technique:** Shift-and-invert; Work with
  
  \[ B = (A - \sigma I)^{-1} \]

- **Some success with polynomial preconditioning** [Chebyshev iteration / least-squares polynomials]. Work with
  
  \[ B = p(A) \]

- **Above preconditioners preserve eigenvectors.** Other methods (Davidson) use a more general preconditioner \( \mathcal{M} \).
Shift-and-invert preconditioning

Main idea: to use Arnoldi, or Lanczos, or subspace iteration for the matrix \( B = (A - \sigma I)^{-1} \). The matrix \( B \) need not be computed explicitly. Each time we need to apply \( B \) to a vector we solve a system with \( B \).

- Factor \( B = A - \sigma I = LU \). Then each solution \( Bx = y \) requires solving \( Lz = y \) and \( Ux = z \).

How to deal with complex shifts?

- If \( A \) is complex need to work in complex arithmetic.
- If \( A \) is real, then instead of \( (A - \sigma I)^{-1} \) use

\[
\Re(A - \sigma I)^{-1} = \frac{1}{2} \left[ (A - \sigma I)^{-1} + (A - \bar{\sigma} I)^{-1} \right]
\]
Preconditioning by polynomials

Main idea:

Iterate with $p(A)$ instead of $A$ in Arnoldi or Lanczos,..

- Used very early on in subspace iteration [Rutishauser, 1959.]
- Usually not as reliable as Shift-and-invert techniques but less demanding in terms of storage.
Question: How to find a good polynomial (dynamically)?

1. Use of Chebyshev polynomials over ellipses
2. Use polynomials based on Leja points
3. Least-squares polynomials over polygons
4. Polynomials from previous Arnoldi decompositions
**Polynomial filters and implicit restart**

Goal: exploit the Arnoldi procedure to apply polynomial filter of the form: \( p(t) = (t - \theta_1)(t - \theta_2) \ldots (t - \theta_q) \)

Assume \( AV_m = V_mH_m + \hat{v}_{m+1}e^T_m \)

and consider first factor: \( (t - \theta_1) \)

\[
(A - \theta_1I)V_m = V_m(H_m - \theta_1I) + \hat{v}_{m+1}e^T_m
\]

Let \( H_m - \theta_1I = Q_1R_1 \). Then,

\[
(A - \theta_1I)V_m = V_mQ_1R_1 + \hat{v}_{m+1}e^T_m \quad \rightarrow \quad (A - \theta_1I)(V_mQ_1) = (V_mQ_1)R_1Q_1 + \hat{v}_{m+1}e^T_mQ_1 \rightarrow \\
A(V_mQ_1) = (V_mQ_1)(R_1Q_1 + \theta_1I) + \hat{v}_{m+1}e^T_mQ_1
\]
Notation:
\[ R_1 Q_1 + \theta_1 I \equiv H_m^{(1)}; \quad (b_{m+1}^{(1)})^T \equiv e_m^T Q_1; \quad V_m Q_1 \equiv V_m^{(1)} \]

\[ AV_m^{(1)} = V_m^{(1)} H_m^{(1)} + v_{m+1} (b_{m+1}^{(1)})^T \]

\[ A V_m^{(1)} = V_m^{(1)} H_m^{(1)} + v_{m+1} (b_{m+1}^{(1)})^T \]

\[ \Rightarrow \text{matrix resulting from one step of the QR algorithm with shift } \theta_1 \text{ applied to } H_m. \]

\[ \Rightarrow \text{First column of } V_m^{(1)} \text{ is a multiple of } (A - \theta_1 I)v_1. \]

\[ \Rightarrow \text{The columns of } V_m^{(1)} \text{ are orthonormal.} \]
Can now apply second shift in same way:

\[(A - \theta_2 I)V_m^{(1)} = V_m^{(1)}(H_m^{(1)} - \theta_2 I) + v_{m+1}(b_{m+1}^{(1)})^T \rightarrow\]

Similar process: \((H_m^{(1)} - \theta_2 I) = Q_2 R_2\) then \(\times Q_2\) to the right:

\[(A - \theta_2 I)V_m^{(1)}Q_2 = (V_m^{(1)}Q_2)(R_2 Q_2) + v_{m+1}(b_{m+1}^{(1)})^T Q_2 \]

\[AV_m^{(2)} = V_m^{(2)}H_m^{(2)} + v_{m+1}(b_{m+1}^{(2)})^T\]

Now:

1st column of \(V_m^{(2)}\) = scalar \(\times (A - \theta_2 I)v_1^{(1)}\)

= scalar \(\times (A - \theta_2 I)(A - \theta_1 I)v_1\)
Note that

\[(b_{m+1}^{(2)})^T = e_m^T Q_1 Q_2 = [0, 0, \ldots, 0, \eta_1, \eta_2, \eta_3]\]

Let: \(\hat{V}_{m-2} = [\hat{v}_1, \ldots, \hat{v}_{m-2}]\) consist of first \(m - 2\) columns of \(V_{m}^{(2)}\) and \(\hat{H}_{m-2} = H_m(1 : m - 2, 1 : m - 2)\). Then

\[A\hat{V}_{m-2} = \hat{V}_{m-2}\hat{H}_{m-2} + \hat{\beta}_{m-1}\hat{v}_{m-1}e_m^T\text{ with }\]

\[\hat{\beta}_{m-1}\hat{v}_{m-1} \equiv \eta_1 v_{m+1} + h_{m-1,m-2}^{(2)} v_{m-1}^{(2)} \parallel \hat{v}_{m-1} \parallel_2 = 1\]

Result: An Arnoldi process of \(m - 2\) steps with the initial vector \(p(A)v_1\).

In other words: We know how to apply polynomial ‘filtering’ via a form of the Arnoldi process, combined with the QR algorithm.