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

Background. New applications in data analytics

➤ 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 λ_i 's with smallest or largest real parts;
- All λ_i's in a certain region of $\mathbb{C}$;
- A few of the dominant eigenvalues;
- All λ_i's (rare).

Large eigenvalue problems in applications

- 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. $\longrightarrow$ Large matrices, *many* eigen-pairs to compute

Background: The main tools

Projection process:

(a) Build a 'good' subspace $K = \mathbf{span}(V)$;

(b) get approximate eigenpairs by a Rayleigh-Ritz process: $\tilde{\lambda}$, $\tilde{u} \in K$ satisfy: $(A - \tilde{\lambda}I)\tilde{u} \perp K \longrightarrow$

$$V^H(A - \tilde{\lambda}I)Vy = 0$$

- $\tilde{\lambda}$ = Ritz value, $\tilde{u} = Vy$ = Ritz vector
- Two common choices for K:
  1) Power subspace $K = \mathbf{span}\{A^k X_0\}$; or $\mathbf{span}\{P_k(A)X_0\}$;
  2) Krylov subspace $K = \mathbf{span}\{v, Av, \cdots, A^{k-1}v\}$

Background. The main tools (cont)

Shift-and-invert:

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

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

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

- Works well for computing *a few* eigenvalues near σ/
- 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).

Background. The main tools (cont)

Deflation:

➤ Once eigenvectors converge remove them from the picture

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 $\boldsymbol{K}$ orthogonal to $\boldsymbol{L}$

➤ Given: Two subspaces $\boldsymbol{K}$ and $\boldsymbol{L}$ of same dimension.

➤ Find: $\tilde{\lambda}, \tilde{u}$ such that

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

Two types of methods:

Orthogonal projection methods: situation when $\boldsymbol{L} = \boldsymbol{K}$.

Oblique projection methods: When $\boldsymbol{L} \neq \boldsymbol{K}$.

Rayleigh-Ritz projection

Given: a subspace $\boldsymbol{X}$ known to contain good approximations to eigenvectors of $\boldsymbol{A}$.
Question: How to extract good approximations to eigenvalues/ eigenvectors from this subspace?

Answer: Rayleigh Ritz process.

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

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

➤ $\boldsymbol{Q}^H\boldsymbol{A}\boldsymbol{Q}\boldsymbol{y} = \tilde{\lambda}\boldsymbol{y}$

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, $C = YRY^H$
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]

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 $B = Z^H AZ$ 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 $S_0 = span\{x_1, x_2, \ldots, x_m\}$ 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 $\mathcal{P}_k$ the orthogonal projector onto the subspace $S_k = span\{X_k\}$. 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

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

where ϵ_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 μ = 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_μ 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 $j = 1, ..., m$ *do*
  Compute $w := Av_j$
  For $i = 1, \ldots, j$, *do* $\begin{cases} h_{i,j} := (w, v_i) \\ w := w - h_{i,j}v_i \end{cases}$
  $h_{j+1,j} = \|w\|_2$; $v_{j+1} = w/h_{j+1,j}$
End

Result of Arnoldi's algorithm

Let

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

1. $V_m = [v_1, v_2, ..., v_m]$ 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. $V_m^T A V_m = H_m \equiv \overline{H}_m -$ last row.

Appliaction to eigenvalue problems

- 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* $u_1^{(m)}$
4. *associated with the rightmost eigenvalue* $\lambda_1^{(m)}$.
5. *If satisfied stop, else set* $v_1 \equiv u_1^{(m)}$ *and goto 2.*

Example:

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

m	$\Re(\lambda)$	$\Im(\lambda)$	Res. Norm
10	0.9987435899D+00	0.0	0.246D-01
20	0.9999523324D+00	0.0	0.144D-02
30	0.1000000368D+01	0.0	0.221D-04
40	0.1000000025D+01	0.0	0.508D-06
50	0.9999999996D+00	0.0	0.138D-07

Restarted Arnoldi (cont.)

➤ 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 ρ_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 $u_1, \ldots, u_n$ 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 λ_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 λ_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 $u_1, u_2, ..$ form a set of Schur vectors for A.

➤ Alternative: implicit deflation (within a procedure such as Arnoldi).

Deflated 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, orthogonaling still against $v_1, ..., v_j$ at step j.

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

Thus, for $k = 2$:

$$V_m = \left[\underbrace{v_1, v_2}_{Locked}, \overbrace{v_3, \ldots, v_m}^{active}\right]$$

$$H_m = \left(\begin{array}{cc|ccc} * & * & * & * & * \\ & * & * & * & * \\ \hline & & * & * & * \\ & & * & * & * \\ & & & * & * \\ & & & & * \end{array}\right)$$

➤ 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 $m = 10$.

m	$\Re e(\lambda)$	$\Im m(\lambda)$	Res. Norm
10	0.9987435899D+00	0.0	0.246D-01
20	0.9999523324D+00	0.0	0.144D-02
30	0.1000000368D+01	0.0	0.221D-04
40	0.1000000025D+01	0.0	0.508D-06
50	0.9999999996D+00	0.0	0.138D-07

- Computing the next 2 eigenvalues of Mark(10).

Eig.	Mat-Vec's	$\Re e(\lambda)$	$\Im m(\lambda)$	Res. Norm
2	60	0.9370509474	0.0	0.870D-03
	69	0.9371549617	0.0	0.175D-04
	78	0.9371501442	0.0	0.313D-06
	87	0.9371501564	0.0	0.490D-08
3	96	0.8112247133	0.0	0.210D-02
	104	0.8097553450	0.0	0.538D-03
	112	0.8096419483	0.0	0.874D-04
	⋮	⋮	⋮	⋮
	⋮	⋮	⋮	⋮
	152	0.8095717167	0.0	0.444D-07

Hermitian case: The Lanczos Algorithm

- The Hessenberg matrix becomes tridiagonal :

$$\boldsymbol{A = A^H} \quad \text{and} \quad \boldsymbol{V_m^H A V_m = H_m} \quad \rightarrow \boldsymbol{H_m = H_m^H}$$

- We can write

$$\boldsymbol{H_m} = \begin{bmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \beta_4 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & & \beta_m & \alpha_m \end{bmatrix} \tag{2}$$

- Consequence: three term recurrence

$$\boxed{\beta_{j+1} v_{j+1} = A v_j - \alpha_j v_j - \beta_j v_{j-1}}$$

ALGORITHM : 4. *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. $\quad w_j := A v_j - \beta_j v_{j-1}$
4. $\quad \alpha_j := (w_j, v_j)$
5. $\quad w_j := w_j - \alpha_j v_j$
6. $\quad \beta_{j+1} := \|w_j\|_2$. *If* $\beta_{j+1} = 0$ *then Stop*
7. $\quad v_{j+1} := w_j / \beta_{j+1}$
8. *EndDo*

Hermitian matrix + Arnoldi $\longrightarrow$ 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 indedependence of the v_is. 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

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

$$\tilde{\lambda}_j = \min_{\left\{ \begin{array}{l} u \in K,\ u \neq 0 \\ u \perp \tilde{u}_1,\ldots,\tilde{u}_{j-1} \end{array} \right.} \frac{(Au,u)}{(u,u)} = \frac{(A\tilde{u}_j,\tilde{u}_j)}{(\tilde{u}_j,\tilde{u}_j)}$$

- Bounds for λ_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

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

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. $\quad \alpha_j = (Av_j, w_j)$
5. $\quad \hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
6. $\quad \hat{w}_{j+1} = A^T w_j - \alpha_j w_j - \delta_j w_{j-1}$
7. $\quad \delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}$. *If $\delta_{j+1} = 0$ Stop*
8. $\quad \beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}$
9. $\quad w_{j+1} = \hat{w}_{j+1}/\beta_{j+1}$
10. $\quad v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}$
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

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

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

$$\begin{aligned} AV_m &= V_m T_m + \delta_{m+1} v_{m+1} e_m^H, \\ A^H W_m &= W_m T_m^H + \bar{\beta}_{m+1} w_{m+1} e_m^H, \\ W_m^H A V_m &= T_m \; . \end{aligned}$$

- If θ_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

Ritz value	Right Ritz vector	Left Ritz vector
θ_j	$V_m y_j$	$W_m z_j$

[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 $\boldsymbol{T}_m$.

Look-ahead Lanczos

Algorithm breaks down when:

$$(\hat{\boldsymbol{v}}_{j+1}, \hat{\boldsymbol{w}}_{j+1}) = 0$$

Three distinct situations.

- 'lucky breakdown' when either $\hat{\boldsymbol{v}}_{j+1}$ or $\hat{\boldsymbol{w}}_{j+1}$ is zero. In this case, eigenvalues of $\boldsymbol{T}_m$ are eigenvalues of $\boldsymbol{A}$.
- $(\hat{\boldsymbol{v}}_{j+1}, \hat{\boldsymbol{w}}_{j+1}) = 0$ but of $\hat{\boldsymbol{v}}_{j+1} \neq 0$, $\hat{\boldsymbol{w}}_{j+1} \neq 0 \rightarrow$ 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 $\boldsymbol{v}_{j+1}, \boldsymbol{w}_{j+1}$ and define $\boldsymbol{v}_{j+2}, \boldsymbol{w}_{j+2}$ from $\boldsymbol{v}_j, \boldsymbol{w}_j$. For example by orthogonalizing $\boldsymbol{A}^2\boldsymbol{v}_j$ ... Can define $\boldsymbol{v}_{j+1}$ somewhat arbitrarily as $\boldsymbol{v}_{j+1} = \boldsymbol{A}\boldsymbol{v}_j$. Similarly for $\boldsymbol{w}_{j+1}$.

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