

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*
- *Text: Chaps. 4 to 8 of:*

https://www-users.cse.umn.edu/~saad/eig_book_2ndEd.pdf

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:

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

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. → 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: $(A - \tilde{\lambda}I)\tilde{u} \perp K \longrightarrow$

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

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

➤ Two common choices for K :

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

2) Krylov subspace $K = \text{span}\{v, Av, \dots, A^{k-1}v\}$

Background. The main tools (cont)

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

Example: power method: $v_j = Av_{j-1}/\text{scaling}$ 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 (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 $Q = [q_1, \dots, q_m]$ an orthonormal basis of X . Then write an approximation in the form $\tilde{u} = Qy$ and obtain y by writing

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

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

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, $(A - \tilde{\lambda}I)u = Qz$ 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 of 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 $X = [x_0, \dots, x_m]$ 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 $Y = [y_1, \dots, y_m]$ 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 $S_0 = \text{span}\{x_1, x_2, \dots, x_m\}$ and assume that S_0 is such that the vectors $\{Px_i\}_{i=1, \dots, m}$ are linearly independent where P is the spectral projector associated with $\lambda_1, \dots, \lambda_m$. Let \mathcal{P}_k the orthogonal projector onto the subspace $S_k = \text{span}\{X_k\}$. Then for each eigenvector u_i of A , $i = 1, \dots, m$, there exists a unique vector s_i in the subspace S_0 such that $Ps_i = u_i$. Moreover, the following inequality is satisfied

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

where ϵ_k tends to zero as k tends to infinity.

Krylov subspace methods

Principle: Projection methods on Krylov subspaces:

$$K_m(A, v_1) = \text{span}\{v_1, Av_1, \dots, A^{m-1}v_1\}$$

- The most important class of iterative methods.
- Many variants exist depending on the subspace L .

Simple properties of K_m [$\mu \equiv \text{deg. of minimal polynomial of } v_1$.]

- $K_m = \{p(A)v_1 \mid p = \text{polynomial of degree } \leq m - 1\}$
- $K_m = K_\mu$ 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 $j = 1, \dots, m$ *do*

Compute $w := Av_j$

For $i = 1, \dots, j$, *do* $\left\{ \begin{array}{l} h_{i,j} := (w, v_i) \\ w := w - h_{i,j}v_i \end{array} \right.$

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

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

Application 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	0.99999999996D+00	0.0	0.138D-07

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, \dots, 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 λ_1, u_1 .
Wielandt deflation considers eigenvalues of

$$A_1 = A - \sigma u_1 v^H$$

Note:

$$\Lambda(A_1) = \{\lambda_1 - \sigma, \lambda_2, \dots, \lambda_n\}$$

Wielandt deflation preserves u_1 as an eigenvector as well as 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 $j = 0 \dots \mu - 1$ Do:
3. Compute a dominant eigenvector of A_j
4. Define $A_{j+1} = A_j - \sigma_j \mathbf{u}_j \mathbf{u}_j^H$
5. End

- Computed $\mathbf{u}_1, \mathbf{u}_2, \dots$ form a set of Schur vectors for A .
- In Arnoldi: Accumulate each new converged eigenvector in columns 1, 2, 3, ... ['locked' set of eigenvectors.] + maintain orthogonality
- Alternative: implicit deflation (within a procedure such as Arnoldi).

Deflated Arnoldi

For $k = 1, \dots, NEV$ do: /* Eigenvalue loop */

1. For $j = k, k + 1, \dots, m$ do: /* Arnoldi loop*/

- Compute $w := Av_j$.
- Orthonormalize w against $v_1, v_2, \dots, v_j \rightarrow v_{j+1}$

2. Compute next approximate eigenpair $\tilde{\lambda}, \tilde{u}$.

3. Orthonormalize \tilde{u} against v_1, \dots, v_j ➤ Result = \tilde{s} = approximate Schur vector.

4. Define $v_k := \tilde{s}$.

5. If approximation not satisfactory go to 1.

6. Else define $h_{i,k} = (Av_k, v_i)$, $i = 1, \dots, k$,

Thus, for $k = 2$:

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

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

➤ Similar techniques in Subspace iteration [G. Stewart's SRRIT]

 Run example with restarted Arnoldi with Deflation in *testArnRD*

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

➤ Continued from earlier example. [First eigenpair by iterative Arnoldi with $m = 10$] We now compute next 2 eigenvalues

Eig.	Mat-Vec's	$\Re(\lambda)$	$\Im(\lambda)$	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 :

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

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

Consequence:

3-term recurrence:

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

Hermitian matrix + Arnoldi \rightarrow Hermitian Lanczos

ALGORITHM : 4. *Lanczos*

1. Choose v_1 of norm unity. Set $\beta_1 \equiv 0, v_0 \equiv 0$
2. For $j = 1, 2, \dots, m$ 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

- 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.

Forms of Re-orthogonalization

Full – reorthogonalize v_{j+1} against all previous v_i 's every time.

Partial – reorthogonalize v_{j+1} against all previous v_i 's only when needed

Selective – reorthogonalize v_{j+1} against computed eigenvectors

None – Do not reorthogonalize - but take measures to deal with 'spurious' eigenvalues.

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

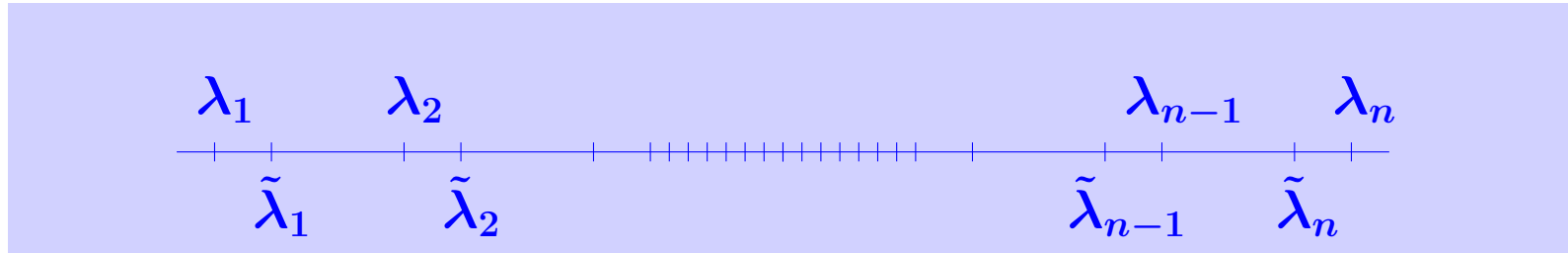
Assume eigenvalues sorted increasingly

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

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

$$\tilde{\lambda}_1 = \min_{u \in K, u \neq 0} \frac{(Au, u)}{(u, u)} = \frac{(A\tilde{u}_1, \tilde{u}_1)}{(\tilde{u}_1, \tilde{u}_1)}$$
$$\tilde{\lambda}_j = \min_{\substack{u \in K, u \neq 0 \\ u \perp \tilde{u}_1, \dots, \tilde{u}_{j-1}}} \frac{(Au, u)}{(u, u)} = \frac{(A\tilde{u}_j, \tilde{u}_j)}{(\tilde{u}_j, \tilde{u}_j)}$$

- Bounds for λ_1 easy to find – similar to linear systems.
- Ritz values approximate eigenvalues of A inside out:



 Run *testLan* to see an illustration

A-priori error bounds

Theorem [Kaniel, 1966]: Let $\gamma_1 = \frac{\lambda_2 - \lambda_1}{\lambda_N - \lambda_2}$; Then:

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

Theorem [Kaniel, Paige, YS]. Let $\gamma_i = \frac{\lambda_{i+1} - \lambda_i}{\lambda_N - \lambda_{i+1}}$, $\kappa_i^{(m)} = \prod_{j < i} \frac{\lambda_j^{(m)} - \lambda_N}{\lambda_j^{(m)} - \lambda_i}$ Then:

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

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 $j = 1, 2, \dots, m$ 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

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

- $v_i \in \mathcal{K}_m(A, v_1)$ and $w_j \in \mathcal{K}_m(A^T, w_1)$.

If the algorithm does not break down before step m , then the vectors $v_i, i = 1, \dots, m$, and $w_j, j = 1, \dots, m$, are biorthogonal, i.e.,

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

Moreover, $\{v_i\}_{i=1,2,\dots,m}$ is a basis of $\mathcal{K}_m(A, v_1)$ and $\{w_i\}_{i=1,2,\dots,m}$ 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 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 T_m .

 Explore the literature on “Look-ahead Lanczos” which aims at resolving some of these issues.