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`

The Problem

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 λ_i 's with smallest or largest real parts;
- Compute all λ_i's in a certain region of $\mathbb{C}$;
- Compute a few of the dominant eigenvalues;
- Compute all λ_i's.

Types of problems

* Standard Hermitian (or symmetric real) $\boxed{Ax = \lambda x}$, $A^H = A$
* Standard non-Hermitian $\boxed{Ax = \lambda x}$, $A^H \neq A$
* 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 Davidosn'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

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

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

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

➤ $Q^H A Q y = \tilde{\lambda} y$

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 $\tilde{U} = QY$

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. $Q^H Q z = 0$ 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 $S_0 = span\{x_1, x_2, \ldots, x_m\}$ 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. $V_m^T AV_m = H_m \equiv \overline{H}_m -$ 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 $k = 2$:

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

$$H_m = \begin{pmatrix} * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & * & * & * \\ & & & * & * \\ & & & & * \end{pmatrix}$$

- 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 $m = 10$.

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 :

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

- We can write

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

- Consequence: three term recurrence

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

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 $\rightarrow$ 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 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. $\alpha_j = (Av_j, w_j)$
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

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

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

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

➤ 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.

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 $A^2 v_j$ ... Can define v_{j+1} somewhat arbitrarily as $v_{j+1} = A v_j$. 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 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

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

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)?

Approaches:

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: $p(t) = (t-\theta_1)(t-\theta_2)\dots(t-\theta_q)$

Assume $AV_m = V_m H_m + \hat{v}_{m+1} e_m^T$
and consider first factor: $(t-\theta_1)$

$$(A-\theta_1 I)V_m = V_m(H_m - \theta_1 I) + \hat{v}_{m+1}e_m^T$$

Let $H_m - \theta_1 I = Q_1 R_1$. Then,

$$\begin{aligned}(A-\theta_1 I)V_m &= V_m Q_1 R_1 + \hat{v}_{m+1}e_m^T \quad \rightarrow \\ (A-\theta_1 I)(V_m Q_1) &= (V_m Q_1)R_1 Q_1 + \hat{v}_{m+1}e_m^T Q_1 \rightarrow \\ A(V_m Q_1) &= (V_m Q_1)(R_1 Q_1 + \theta_1 I) \\ &\quad + \hat{v}_{m+1}e_m^T Q_1\end{aligned}$$

Notation:
$R_1 Q_1 + \theta_1 I \equiv H_m^{(1)}; \quad (b_{m+1}^{(1)})^T \equiv e_m^T Q_1; \quad V_m Q_1 \equiv V_m^{(1)}$

- $AV_m^{(1)} = V_m^{(1)} H_m^{(1)} + v_{m+1}(b_{m+1}^{(1)})^T$
- Note that $H_m^{(1)}$ is upper Hessenberg.
- Similar to an Arnoldi decomposition.

Observe:

- $R_1 Q_1 + \theta_1 I \equiv$ matrix resulting from one step of the QR algorithm with shift θ_1 applied to H_m.
- First column of $V_m^{(1)}$ is a multiple of $(A-\theta_1 I)v_1$.
- The columns of $V_m^{(1)}$ are orthonormal.

Can now apply second shift in same way:

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

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

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

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

$$\begin{aligned}A\hat{V}_{m-2} &= \hat{V}_{m-2}\hat{H}_{m-2} + \hat{\beta}_{m-1}\hat{v}_{m-1}e_m^T \quad \text{with} \\ \hat{\beta}_{m-1}\hat{v}_{m-1} &\equiv \eta_1 v_{m+1} + h_{m-1,m-2}^{(2)} v_{m-1}^{(2)} \quad \|\hat{v}_{m-1}\|_2 = 1\end{aligned}$$

- 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.