Scientific Computing: An Introductory Survey

Chapter 4 – Eigenvalue Problems

Prof. Michael T. Heath

Department of Computer Science
University of Illinois at Urbana-Champaign

Outline

Eigenvalue Problems

- Eigenvalue problems occur in many areas of science and engineering, such as structural analysis
- Eigenvalues are also important in analyzing numerical methods
- Theory and algorithms apply to complex matrices as well as real matrices
- With complex matrices, we use conjugate transpose, $\boldsymbol{A}^H$, instead of usual transpose, $\boldsymbol{A}^T$

Eigenvalues and Eigenvectors

- Standard *eigenvalue problem*: Given $n \times n$ matrix $\boldsymbol{A}$, find scalar λ and nonzero vector $\boldsymbol{x}$ such that

$$\boldsymbol{A}\,\boldsymbol{x} = \lambda\,\boldsymbol{x}$$

- λ is *eigenvalue*, and $\boldsymbol{x}$ is corresponding *eigenvector*
- λ may be complex even if $\boldsymbol{A}$ is real
- *Spectrum* $= \lambda(\boldsymbol{A}) =$ set of eigenvalues of $\boldsymbol{A}$
- *Spectral radius* $= \rho(\boldsymbol{A}) = \max\{|\lambda| \,:\, \lambda \in \lambda(\boldsymbol{A})\}$

Geometric Interpretation

- Matrix expands or shrinks any vector lying in direction of eigenvector by scalar factor
- Expansion or contraction factor is given by corresponding eigenvalue λ
- Eigenvalues and eigenvectors decompose complicated behavior of general linear transformation into simpler actions

Examples: Eigenvalues and Eigenvectors

- $\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$: $\lambda_1 = 1,\ \boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \lambda_2 = 2,\ \boldsymbol{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
- $\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$: $\lambda_1 = 1,\ \boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \lambda_2 = 2,\ \boldsymbol{x}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
- $\boldsymbol{A} = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}$: $\lambda_1 = 2,\ \boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \lambda_2 = 4,\ \boldsymbol{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$
- $\boldsymbol{A} = \begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix}$: $\lambda_1 = 2,\ \boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \lambda_2 = 1,\ \boldsymbol{x}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$
- $\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$: $\lambda_1 = i,\ \boldsymbol{x}_1 = \begin{bmatrix} 1 \\ i \end{bmatrix}, \quad \lambda_2 = -i,\ \boldsymbol{x}_2 = \begin{bmatrix} i \\ 1 \end{bmatrix}$

  where $i = \sqrt{-1}$

Characteristic Polynomial

- Equation $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$ is equivalent to

$$(\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{x} = \boldsymbol{0}$$

  which has nonzero solution $\boldsymbol{x}$ if, and only if, its matrix is singular
- Eigenvalues of $\boldsymbol{A}$ are roots λ_i of *characteristic polynomial*

$$\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0$$

  in λ of degree n
- *Fundamental Theorem of Algebra* implies that $n \times n$ matrix $\boldsymbol{A}$ always has n eigenvalues, but they may not be real nor distinct
- Complex eigenvalues of real matrix occur in complex conjugate pairs: if $\alpha + i\beta$ is eigenvalue of real matrix, then so is $\alpha - i\beta$, where $i = \sqrt{-1}$

Example: Characteristic Polynomial

- Characteristic polynomial of previous example matrix is

$$\det\left(\begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) =$$

$$\det\left(\begin{bmatrix} 3-\lambda & -1 \\ -1 & 3-\lambda \end{bmatrix}\right) =$$

$$(3-\lambda)(3-\lambda) - (-1)(-1) = \lambda^2 - 6\lambda + 8 = 0$$

so eigenvalues are given by

$$\lambda = \frac{6 \pm \sqrt{36-32}}{2}, \quad \text{or} \quad \lambda_1 = 2, \quad \lambda_2 = 4$$

Companion Matrix

- Monic polynomial

$$p(\lambda) = c_0 + c_1\lambda + \cdots + c_{n-1}\lambda^{n-1} + \lambda^n$$

is characteristic polynomial of *companion matrix*

$$\boldsymbol{C}_n = \begin{bmatrix} 0 & 0 & \cdots & 0 & -c_0 \\ 1 & 0 & \cdots & 0 & -c_1 \\ 0 & 1 & \cdots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -c_{n-1} \end{bmatrix}$$

- Roots of polynomial of degree > 4 cannot always computed in finite number of steps
- So in general, computation of eigenvalues of matrices of order > 4 requires (theoretically infinite) iterative process

Characteristic Polynomial, continued

- Computing eigenvalues using characteristic polynomial is *not* recommended because of
  - work in computing coefficients of characteristic polynomial
  - sensitivity of coefficients of characteristic polynomial
  - work in solving for roots of characteristic polynomial
- Characteristic polynomial is powerful theoretical tool but usually not useful computationally

Example: Characteristic Polynomial

- Consider

$$\boldsymbol{A} = \begin{bmatrix} 1 & \epsilon \\ \epsilon & 1 \end{bmatrix}$$

  where ϵ is positive number slightly smaller than $\sqrt{\epsilon_{\text{mach}}}$
- Exact eigenvalues of $\boldsymbol{A}$ are $1 + \epsilon$ and $1 - \epsilon$
- Computing characteristic polynomial in floating-point arithmetic, we obtain

$$\det(\boldsymbol{A} - \lambda \boldsymbol{I}) = \lambda^2 - 2\lambda + (1 - \epsilon^2) = \lambda^2 - 2\lambda + 1$$

  which has 1 as double root
- Thus, eigenvalues cannot be resolved by this method even though they are distinct in working precision

Multiplicity and Diagonalizability

- *Multiplicity* is number of times root appears when polynomial is written as product of linear factors
- Eigenvalue of multiplicity 1 is *simple*
- *Defective* matrix has eigenvalue of multiplicity $k > 1$ with fewer than k linearly independent corresponding eigenvectors
- Nondefective matrix $\boldsymbol{A}$ has n linearly independent eigenvectors, so it is *diagonalizable*

$$\boldsymbol{X}^{-1}\boldsymbol{A}\boldsymbol{X} = \boldsymbol{D}$$

where $\boldsymbol{X}$ is nonsingular matrix of eigenvectors

Eigenspaces and Invariant Subspaces

- Eigenvectors can be scaled arbitrarily: if $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$, then $\boldsymbol{A}(\gamma\boldsymbol{x}) = \lambda(\gamma\boldsymbol{x})$ for any scalar γ, so $\gamma\boldsymbol{x}$ is also eigenvector corresponding to λ
- Eigenvectors are usually *normalized* by requiring some norm of eigenvector to be 1
- *Eigenspace* $= \mathcal{S}_\lambda = \{\boldsymbol{x} \,:\, \boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}\}$
- Subspace $\mathcal{S}$ of $\mathbb{R}^n$ (or $\mathbb{C}^n$) is *invariant* if $\boldsymbol{A}\mathcal{S} \subseteq \mathcal{S}$
- For eigenvectors $\boldsymbol{x}_1 \cdots \boldsymbol{x}_p$, span$([\boldsymbol{x}_1 \cdots \boldsymbol{x}_p])$ is invariant subspace

Relevant Properties of Matrices

- Properties of matrix $\boldsymbol{A}$ relevant to eigenvalue problems

Property	Definition
diagonal	$a_{ij} = 0$ for $i \neq j$
tridiagonal	$a_{ij} = 0$ for $\lvert i - j \rvert > 1$
triangular	$a_{ij} = 0$ for $i > j$ (upper)
	$a_{ij} = 0$ for $i < j$ (lower)
Hessenberg	$a_{ij} = 0$ for $i > j + 1$ (upper)
	$a_{ij} = 0$ for $i < j - 1$ (lower)
orthogonal	$\boldsymbol{A}^T \boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{I}$
unitary	$\boldsymbol{A}^H \boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^H = \boldsymbol{I}$
symmetric	$\boldsymbol{A} = \boldsymbol{A}^T$
Hermitian	$\boldsymbol{A} = \boldsymbol{A}^H$
normal	$\boldsymbol{A}^H \boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^H$

Examples: Matrix Properties

- Transpose: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$
- Conjugate transpose: $\begin{bmatrix} 1+i & 1+2i \\ 2-i & 2-2i \end{bmatrix}^H = \begin{bmatrix} 1-i & 2+i \\ 1-2i & 2+2i \end{bmatrix}$
- Symmetric: $\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$
- Nonsymmetric: $\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$
- Hermitian: $\begin{bmatrix} 1 & 1+i \\ 1-i & 2 \end{bmatrix}$
- NonHermitian: $\begin{bmatrix} 1 & 1+i \\ 1+i & 2 \end{bmatrix}$

Examples, continued

- Orthogonal: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 \\ -\sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}$
- Unitary: $\begin{bmatrix} i\sqrt{2}/2 & \sqrt{2}/2 \\ -\sqrt{2}/2 & -i\sqrt{2}/2 \end{bmatrix}$
- Nonorthogonal: $\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$
- Normal: $\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix}$
- Nonnormal: $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$

Properties of Eigenvalue Problems

Properties of eigenvalue problem affecting choice of algorithm and software

- Are all eigenvalues needed, or only a few?
- Are only eigenvalues needed, or are corresponding eigenvectors also needed?
- Is matrix real or complex?
- Is matrix relatively small and dense, or large and sparse?
- Does matrix have any special properties, such as symmetry, or is it general matrix?

Conditioning of Eigenvalue Problems

- Condition of eigenvalue problem is sensitivity of eigenvalues and eigenvectors to changes in matrix
- Conditioning of eigenvalue problem is *not* same as conditioning of solution to linear system for same matrix
- Different eigenvalues and eigenvectors are not necessarily equally sensitive to perturbations in matrix

Conditioning of Eigenvalues

- If μ is eigenvalue of perturbation $\boldsymbol{A} + \boldsymbol{E}$ of nondefective matrix $\boldsymbol{A}$, then

$$|\mu - \lambda_k| \leq \mathrm{cond}_2(\boldsymbol{X}) \, \|\boldsymbol{E}\|_2$$

  where λ_k is closest eigenvalue of $\boldsymbol{A}$ to μ and $\boldsymbol{X}$ is nonsingular matrix of eigenvectors of $\boldsymbol{A}$
- Absolute condition number of eigenvalues is condition number of matrix of eigenvectors with respect to solving linear equations
- Eigenvalues may be sensitive if eigenvectors are nearly linearly dependent (i.e., matrix is nearly defective)
- For *normal* matrix ($\boldsymbol{A}^H \boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^H$), eigenvectors are orthogonal, so eigenvalues are well-conditioned

Conditioning of Eigenvalues

- If $(\boldsymbol{A}+\boldsymbol{E})(\boldsymbol{x}+\Delta\boldsymbol{x}) = (\lambda+\Delta\lambda)(\boldsymbol{x}+\Delta\boldsymbol{x})$, where λ is simple eigenvalue of $\boldsymbol{A}$, then

$$|\Delta\lambda| \lessapprox \frac{\|\boldsymbol{y}\|_2 \cdot \|\boldsymbol{x}\|_2}{|\boldsymbol{y}^H\boldsymbol{x}|}\,\|\boldsymbol{E}\|_2 = \frac{1}{\cos(\theta)}\,\|\boldsymbol{E}\|_2$$

  where $\boldsymbol{x}$ and $\boldsymbol{y}$ are corresponding right and left eigenvectors and θ is angle between them
- For symmetric or Hermitian matrix, right and left eigenvectors are same, so $\cos(\theta) = 1$ and eigenvalues are inherently well-conditioned
- Eigenvalues of nonnormal matrices may be sensitive
- For multiple or closely clustered eigenvalues, corresponding eigenvectors may be sensitive

Problem Transformations

- *Shift*: If $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$ and σ is any scalar, then $(\boldsymbol{A} - \sigma\boldsymbol{I})\boldsymbol{x} = (\lambda - \sigma)\boldsymbol{x}$, so eigenvalues of shifted matrix are shifted eigenvalues of original matrix
- *Inversion*: If $\boldsymbol{A}$ is nonsingular and $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$ with $\boldsymbol{x} \neq \boldsymbol{0}$, then $\lambda \neq 0$ and $\boldsymbol{A}^{-1}\boldsymbol{x} = (1/\lambda)\boldsymbol{x}$, so eigenvalues of inverse are reciprocals of eigenvalues of original matrix
- *Powers*: If $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$, then $\boldsymbol{A}^k\boldsymbol{x} = \lambda^k\boldsymbol{x}$, so eigenvalues of power of matrix are same power of eigenvalues of original matrix
- *Polynomial*: If $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$ and $p(t)$ is polynomial, then $p(\boldsymbol{A})\boldsymbol{x} = p(\lambda)\boldsymbol{x}$, so eigenvalues of polynomial in matrix are values of polynomial evaluated at eigenvalues of original matrix

Similarity Transformation

- B is *similar* to A if there is nonsingular matrix T such that

$$B = T^{-1} A\, T$$

- Then

$$By = \lambda y \;\Rightarrow\; T^{-1}ATy = \lambda y \;\Rightarrow\; A(Ty) = \lambda(Ty)$$

  so A and B have same eigenvalues, and if y is eigenvector of B, then $x = Ty$ is eigenvector of A

- Similarity transformations preserve eigenvalues and eigenvectors are easily recovered

Example: Similarity Transformation

- From eigenvalues and eigenvectors for previous example,

$$\begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

  and hence

$$\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

- So original matrix is similar to diagonal matrix, and eigenvectors form columns of similarity transformation matrix

Diagonal Form

- Eigenvalues of diagonal matrix are diagonal entries, and eigenvectors are columns of identity matrix
- Diagonal form is desirable in simplifying eigenvalue problems for general matrices by similarity transformations
- But not all matrices are diagonalizable by similarity transformation
- Closest one can get, in general, is *Jordan form*, which is nearly diagonal but may have some nonzero entries on first superdiagonal, corresponding to one or more multiple eigenvalues

Triangular Form

- Any matrix can be transformed into triangular (*Schur*) form by similarity, and eigenvalues of triangular matrix are diagonal entries
- Eigenvectors of triangular matrix less obvious, but still straightforward to compute
- If

$$\boldsymbol{A} - \lambda \boldsymbol{I} = \begin{bmatrix} \boldsymbol{U}_{11} & \boldsymbol{u} & \boldsymbol{U}_{13} \\ \boldsymbol{0} & 0 & \boldsymbol{v}^T \\ \boldsymbol{O} & \boldsymbol{0} & \boldsymbol{U}_{33} \end{bmatrix}$$

is triangular, then $\boldsymbol{U}_{11}\boldsymbol{y} = \boldsymbol{u}$ can be solved for $\boldsymbol{y}$, so that

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{y} \\ -1 \\ \boldsymbol{0} \end{bmatrix}$$

is corresponding eigenvector

Block Triangular Form

- If

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \cdots & \boldsymbol{A}_{1p} \\ & \boldsymbol{A}_{22} & \cdots & \boldsymbol{A}_{2p} \\ & & \ddots & \vdots \\ & & & \boldsymbol{A}_{pp} \end{bmatrix}$$

  with *square* diagonal blocks, then

$$\lambda(\boldsymbol{A}) = \bigcup_{j=1}^{p} \lambda(\boldsymbol{A}_{jj})$$

  so eigenvalue problem breaks into p smaller eigenvalue problems
- *Real* Schur form has 1×1 diagonal blocks corresponding to real eigenvalues and 2×2 diagonal blocks corresponding to pairs of complex conjugate eigenvalues

Forms Attainable by Similarity

A	T	B
distinct eigenvalues	nonsingular	diagonal
real symmetric	orthogonal	real diagonal
complex Hermitian	unitary	real diagonal
normal	unitary	diagonal
arbitrary real	orthogonal	real block triangular (real Schur)
arbitrary	unitary	upper triangular (Schur)
arbitrary	nonsingular	almost diagonal (Jordan)

- Given matrix A with indicated property, matrices B and T exist with indicated properties such that $B = T^{-1}AT$
- If B is diagonal or triangular, eigenvalues are its diagonal entries
- If B is diagonal, eigenvectors are columns of T

Power Iteration

- Simplest method for computing one eigenvalue-eigenvector pair is *power iteration*, which repeatedly multiplies matrix times initial starting vector
- Assume $\boldsymbol{A}$ has unique eigenvalue of maximum modulus, say λ_1, with corresponding eigenvector $\boldsymbol{v}_1$
- Then, starting from nonzero vector $\boldsymbol{x}_0$, iteration scheme

$$\boldsymbol{x}_k = \boldsymbol{A}\boldsymbol{x}_{k-1}$$

converges to multiple of eigenvector $\boldsymbol{v}_1$ corresponding to *dominant* eigenvalue λ_1

Convergence of Power Iteration

- To see why power iteration converges to dominant eigenvector, express starting vector $\boldsymbol{x}_0$ as linear combination

$$\boldsymbol{x}_0 = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i$$

  where $\boldsymbol{v}_i$ are eigenvectors of $\boldsymbol{A}$
- Then

$$\boldsymbol{x}_k = \boldsymbol{A}\boldsymbol{x}_{k-1} = \boldsymbol{A}^2\boldsymbol{x}_{k-2} = \cdots = \boldsymbol{A}^k\boldsymbol{x}_0 =$$

$$\sum_{i=1}^{n} \lambda_i^k \alpha_i \boldsymbol{v}_i = \lambda_1^k \left(\alpha_1 \boldsymbol{v}_1 + \sum_{i=2}^{n} (\lambda_i/\lambda_1)^k \alpha_i \boldsymbol{v}_i \right)$$

- Since $|\lambda_i/\lambda_1| < 1$ for $i > 1$, successively higher powers go to zero, leaving only component corresponding to $\boldsymbol{v}_1$

Example: Power Iteration

- Ratio of values of given component of $\boldsymbol{x}_k$ from one iteration to next converges to dominant eigenvalue λ_1
- For example, if $\boldsymbol{A} = \begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix}$ and $\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, we obtain

k	$\boldsymbol{x}_k^T$		ratio
0	0.0	1.0	
1	0.5	1.5	1.500
2	1.5	2.5	1.667
3	3.5	4.5	1.800
4	7.5	8.5	1.889
5	15.5	16.5	1.941
6	31.5	32.5	1.970
7	63.5	64.5	1.985
8	127.5	128.5	1.992

- Ratio is converging to dominant eigenvalue, which is 2

Limitations of Power Iteration

Power iteration can fail for various reasons

- Starting vector may have *no* component in dominant eigenvector $\boldsymbol{v}_1$ (i.e., $\alpha_1 = 0$) — not problem in practice because rounding error usually introduces such component in any case
- There may be more than one eigenvalue having same (maximum) modulus, in which case iteration may converge to linear combination of corresponding eigenvectors
- For real matrix and starting vector, iteration can never converge to complex vector

Normalized Power Iteration

- Geometric growth of components at each iteration risks eventual overflow (or underflow if $\lambda_1 < 1$)
- Approximate eigenvector should be normalized at each iteration, say, by requiring its largest component to be 1 in modulus, giving iteration scheme

$$\begin{aligned} \boldsymbol{y}_k &= \boldsymbol{A}\boldsymbol{x}_{k-1} \\ \boldsymbol{x}_k &= \boldsymbol{y}_k / \|\boldsymbol{y}_k\|_\infty \end{aligned}$$

- With normalization, $\|\boldsymbol{y}_k\|_\infty \to |\lambda_1|$, and $\boldsymbol{x}_k \to \boldsymbol{v}_1 / \|\boldsymbol{v}_1\|_\infty$

Example: Normalized Power Iteration

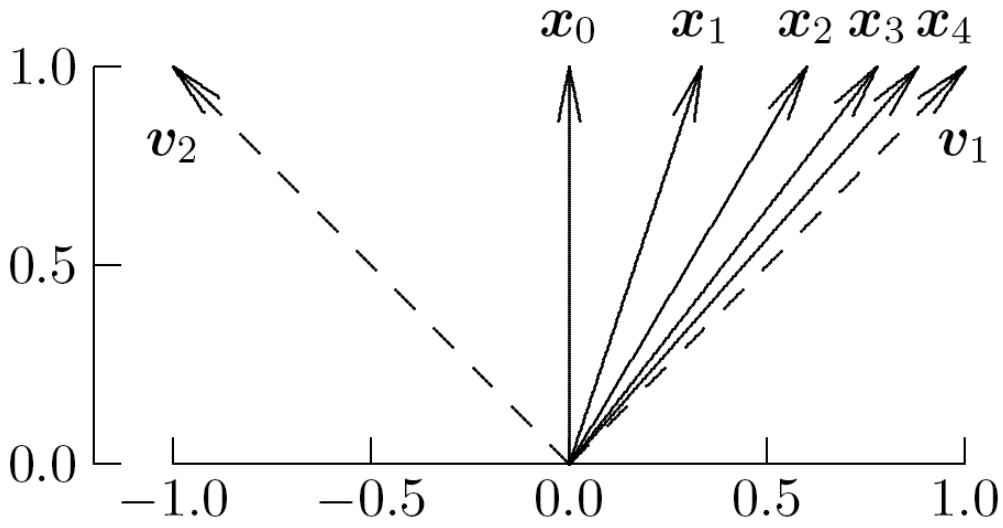
- Repeating previous example with normalized scheme,

k	$\boldsymbol{x}_k^T$		$\|\boldsymbol{y}_k\|_\infty$
0	0.000	1.0	
1	0.333	1.0	1.500
2	0.600	1.0	1.667
3	0.778	1.0	1.800
4	0.882	1.0	1.889
5	0.939	1.0	1.941
6	0.969	1.0	1.970
7	0.984	1.0	1.985
8	0.992	1.0	1.992

< interactive example >

Geometric Interpretation

- Behavior of power iteration depicted geometrically



- Initial vector $x_0 = v_1 + v_2$ contains equal components in eigenvectors v_1 and v_2 (dashed arrows)
- Repeated multiplication by A causes component in v_1 (corresponding to larger eigenvalue, 2) to dominate, so sequence of vectors x_k converges to v_1

Power Iteration with Shift

- Convergence rate of power iteration depends on ratio $|\lambda_2/\lambda_1|$, where λ_2 is eigenvalue having second largest modulus
- May be possible to choose shift, $\boldsymbol{A} - \sigma \boldsymbol{I}$, such that

$$\left|\frac{\lambda_2 - \sigma}{\lambda_1 - \sigma}\right| < \left|\frac{\lambda_2}{\lambda_1}\right|$$

  so convergence is accelerated
- Shift must then be added to result to obtain eigenvalue of original matrix

Example: Power Iteration with Shift

- In earlier example, for instance, if we pick shift of $\sigma = 1$, (which is equal to other eigenvalue) then ratio becomes zero and method converges in one iteration
- In general, we would not be able to make such fortuitous choice, but shifts can still be extremely useful in some contexts, as we will see later

Inverse Iteration

- If smallest eigenvalue of matrix required rather than largest, can make use of fact that eigenvalues of $\boldsymbol{A}^{-1}$ are reciprocals of those of $\boldsymbol{A}$, so smallest eigenvalue of $\boldsymbol{A}$ is reciprocal of largest eigenvalue of $\boldsymbol{A}^{-1}$
- This leads to *inverse iteration* scheme

$$\begin{aligned} \boldsymbol{A}\boldsymbol{y}_k &= \boldsymbol{x}_{k-1} \\ \boldsymbol{x}_k &= \boldsymbol{y}_k / \|\boldsymbol{y}_k\|_\infty \end{aligned}$$

  which is equivalent to power iteration applied to $\boldsymbol{A}^{-1}$
- Inverse of $\boldsymbol{A}$ not computed explicitly, but factorization of $\boldsymbol{A}$ used to solve system of linear equations at each iteration

Inverse Iteration, continued

- Inverse iteration converges to eigenvector corresponding to *smallest* eigenvalue of $\boldsymbol{A}$
- Eigenvalue obtained is dominant eigenvalue of $\boldsymbol{A}^{-1}$, and hence its reciprocal is smallest eigenvalue of $\boldsymbol{A}$ in modulus

Example: Inverse Iteration

- Applying inverse iteration to previous example to compute smallest eigenvalue yields sequence

k	$\boldsymbol{x}_k^T$		$\|\boldsymbol{y}_k\|_\infty$
0	0.000	1.0	
1	−0.333	1.0	0.750
2	−0.600	1.0	0.833
3	−0.778	1.0	0.900
4	−0.882	1.0	0.944
5	−0.939	1.0	0.971
6	−0.969	1.0	0.985

which is indeed converging to 1 (which is its own reciprocal in this case)

< interactive example >

Inverse Iteration with Shift

- As before, shifting strategy, working with $\boldsymbol{A} - \sigma \boldsymbol{I}$ for some scalar σ, can greatly improve convergence
- Inverse iteration is particularly useful for computing eigenvector corresponding to approximate eigenvalue, since it converges rapidly when applied to shifted matrix $\boldsymbol{A} - \lambda \boldsymbol{I}$, where λ is approximate eigenvalue
- Inverse iteration is also useful for computing eigenvalue closest to given value β, since if β is used as shift, then desired eigenvalue corresponds to smallest eigenvalue of shifted matrix

Rayleigh Quotient

- Given approximate eigenvector $\boldsymbol{x}$ for real matrix $\boldsymbol{A}$, determining best estimate for corresponding eigenvalue λ can be considered as $n \times 1$ linear least squares approximation problem

$$\boldsymbol{x}\lambda \cong \boldsymbol{A}\boldsymbol{x}$$

- From normal equation $\boldsymbol{x}^T\boldsymbol{x}\lambda = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$, least squares solution is given by

$$\lambda = \frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}$$

- This quantity, known as *Rayleigh quotient*, has many useful properties

Example: Rayleigh Quotient

- Rayleigh quotient can accelerate convergence of iterative methods such as power iteration, since Rayleigh quotient $\boldsymbol{x}_k^T \boldsymbol{A} \boldsymbol{x}_k / \boldsymbol{x}_k^T \boldsymbol{x}_k$ gives better approximation to eigenvalue at iteration k than does basic method alone
- For previous example using power iteration, value of Rayleigh quotient at each iteration is shown below

k	$\boldsymbol{x}_k^T$		$\|\boldsymbol{y}_k\|_\infty$	$\boldsymbol{x}_k^T \boldsymbol{A} \boldsymbol{x}_k / \boldsymbol{x}_k^T \boldsymbol{x}_k$
0	0.000	1.0		
1	0.333	1.0	1.500	1.500
2	0.600	1.0	1.667	1.800
3	0.778	1.0	1.800	1.941
4	0.882	1.0	1.889	1.985
5	0.939	1.0	1.941	1.996
6	0.969	1.0	1.970	1.999

Rayleigh Quotient Iteration

- Given approximate eigenvector, Rayleigh quotient yields good estimate for corresponding eigenvalue
- Conversely, inverse iteration converges rapidly to eigenvector if approximate eigenvalue is used as shift, with one iteration often sufficing
- These two ideas combined in *Rayleigh quotient iteration*

$$\sigma_k = \boldsymbol{x}_k^T \boldsymbol{A} \boldsymbol{x}_k / \boldsymbol{x}_k^T \boldsymbol{x}_k$$

$$(\boldsymbol{A} - \sigma_k \boldsymbol{I}) \boldsymbol{y}_{k+1} = \boldsymbol{x}_k$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1} / \|\boldsymbol{y}_{k+1}\|_\infty$$

starting from given nonzero vector $\boldsymbol{x}_0$

Rayleigh Quotient Iteration, continued

- Rayleigh quotient iteration is especially effective for symmetric matrices and usually converges very rapidly
- Using different shift at each iteration means matrix must be refactored each time to solve linear system, so cost per iteration is high unless matrix has special form that makes factorization easy
- Same idea also works for complex matrices, for which transpose is replaced by conjugate transpose, so Rayleigh quotient becomes $\boldsymbol{x}^H \boldsymbol{A}\boldsymbol{x}/\boldsymbol{x}^H\boldsymbol{x}$

Example: Rayleigh Quotient Iteration

- Using same matrix as previous examples and randomly chosen starting vector $\boldsymbol{x}_0$, Rayleigh quotient iteration converges in two iterations

k	$\boldsymbol{x}_k^T$		σ_k
0	0.807	0.397	1.896
1	0.924	1.000	1.998
2	1.000	1.000	2.000

Deflation

- After eigenvalue λ_1 and corresponding eigenvector $\boldsymbol{x}_1$ have been computed, then additional eigenvalues $\lambda_2, \ldots, \lambda_n$ of $\boldsymbol{A}$ can be computed by *deflation*, which effectively removes known eigenvalue
- Let $\boldsymbol{H}$ be any nonsingular matrix such that $\boldsymbol{H}\boldsymbol{x}_1 = \alpha \boldsymbol{e}_1$, scalar multiple of first column of identity matrix (Householder transformation is good choice for $\boldsymbol{H}$)
- Then similarity transformation determined by $\boldsymbol{H}$ transforms $\boldsymbol{A}$ into form

$$\boldsymbol{H}\boldsymbol{A}\boldsymbol{H}^{-1} = \begin{bmatrix} \lambda_1 & \boldsymbol{b}^T \\ \boldsymbol{0} & \boldsymbol{B} \end{bmatrix}$$

where $\boldsymbol{B}$ is matrix of order $n-1$ having eigenvalues $\lambda_2, \ldots, \lambda_n$

Deflation, continued

- Thus, we can work with $\boldsymbol{B}$ to compute next eigenvalue λ_2
- Moreover, if $\boldsymbol{y}_2$ is eigenvector of $\boldsymbol{B}$ corresponding to λ_2, then

$$\boldsymbol{x}_2 = \boldsymbol{H}^{-1} \begin{bmatrix} \alpha \\ \boldsymbol{y}_2 \end{bmatrix}, \quad \text{where} \quad \alpha = \frac{\boldsymbol{b}^T \boldsymbol{y}_2}{\lambda_2 - \lambda_1}$$

  is eigenvector corresponding to λ_2 for original matrix $\boldsymbol{A}$, provided $\lambda_1 \neq \lambda_2$
- Process can be repeated to find additional eigenvalues and eigenvectors

Deflation, continued

- Alternative approach lets $\boldsymbol{u}_1$ be any vector such that $\boldsymbol{u}_1^T \boldsymbol{x}_1 = \lambda_1$
- Then $\boldsymbol{A} - \boldsymbol{x}_1 \boldsymbol{u}_1^T$ has eigenvalues $0, \lambda_2, \ldots, \lambda_n$
- Possible choices for $\boldsymbol{u}_1$ include
  - $\boldsymbol{u}_1 = \lambda_1 \boldsymbol{x}_1$, if $\boldsymbol{A}$ is symmetric and $\boldsymbol{x}_1$ is normalized so that $\|\boldsymbol{x}_1\|_2 = 1$
  - $\boldsymbol{u}_1 = \lambda_1 \boldsymbol{y}_1$, where $\boldsymbol{y}_1$ is corresponding left eigenvector (i.e., $\boldsymbol{A}^T \boldsymbol{y}_1 = \lambda_1 \boldsymbol{y}_1$) normalized so that $\boldsymbol{y}_1^T \boldsymbol{x}_1 = 1$
  - $\boldsymbol{u}_1 = \boldsymbol{A}^T \boldsymbol{e}_k$, if $\boldsymbol{x}_1$ is normalized so that $\|\boldsymbol{x}_1\|_\infty = 1$ and kth component of $\boldsymbol{x}_1$ is 1

Simultaneous Iteration

- Simplest method for computing many eigenvalue-eigenvector pairs is *simultaneous iteration*, which repeatedly multiplies matrix times matrix of initial starting vectors
- Starting from $n \times p$ matrix $\boldsymbol{X}_0$ of rank p, iteration scheme is

$$\boldsymbol{X}_k = \boldsymbol{A}\boldsymbol{X}_{k-1}$$

- span$(\boldsymbol{X}_k)$ converges to invariant subspace determined by p largest eigenvalues of $\boldsymbol{A}$, provided $|\lambda_p| > |\lambda_{p+1}|$
- Also called *subspace iteration*

Orthogonal Iteration

- As with power iteration, normalization is needed with simultaneous iteration
- Each column of $\boldsymbol{X}_k$ converges to dominant eigenvector, so columns of $\boldsymbol{X}_k$ become increasingly ill-conditioned basis for span$(\boldsymbol{X}_k)$
- Both issues can be addressed by computing QR factorization at each iteration

$$\begin{aligned} \hat{\boldsymbol{Q}}_k \boldsymbol{R}_k &= \boldsymbol{X}_{k-1} \\ \boldsymbol{X}_k &= \boldsymbol{A}\hat{\boldsymbol{Q}}_k \end{aligned}$$

  where $\hat{\boldsymbol{Q}}_k \boldsymbol{R}_k$ is *reduced* QR factorization of $\boldsymbol{X}_{k-1}$
- This *orthogonal iteration* converges to block triangular form, and leading block is triangular if moduli of consecutive eigenvalues are distinct

QR Iteration

- For $p = n$ and $\boldsymbol{X}_0 = \boldsymbol{I}$, matrices

$$\boldsymbol{A}_k = \hat{\boldsymbol{Q}}_k^H \boldsymbol{A} \hat{\boldsymbol{Q}}_k$$

  generated by orthogonal iteration converge to triangular or block triangular form, yielding all eigenvalues of $\boldsymbol{A}$

- *QR iteration* computes successive matrices $\boldsymbol{A}_k$ without forming above product explicitly

- Starting with $\boldsymbol{A}_0 = \boldsymbol{A}$, at iteration k compute QR factorization

$$\boldsymbol{Q}_k \boldsymbol{R}_k = \boldsymbol{A}_{k-1}$$

  and form reverse product

$$\boldsymbol{A}_k = \boldsymbol{R}_k \boldsymbol{Q}_k$$

QR Iteration, continued

- Successive matrices $\boldsymbol{A}_k$ are unitarily similar to each other

$$\boldsymbol{A}_k = \boldsymbol{R}_k \boldsymbol{Q}_k = \boldsymbol{Q}_k^H \boldsymbol{A}_{k-1} \boldsymbol{Q}_k$$

- Diagonal entries (or eigenvalues of diagonal blocks) of $\boldsymbol{A}_k$ converge to eigenvalues of $\boldsymbol{A}$
- Product of orthogonal matrices $\boldsymbol{Q}_k$ converges to matrix of corresponding eigenvectors
- If $\boldsymbol{A}$ is symmetric, then symmetry is preserved by QR iteration, so $\boldsymbol{A}_k$ converge to matrix that is both triangular and symmetric, hence diagonal

Example: QR Iteration

- Let $\boldsymbol{A}_0 = \begin{bmatrix} 7 & 2 \\ 2 & 4 \end{bmatrix}$
- Compute QR factorization

$$\boldsymbol{A}_0 = \boldsymbol{Q}_1 \boldsymbol{R}_1 = \begin{bmatrix} .962 & -.275 \\ .275 & .962 \end{bmatrix} \begin{bmatrix} 7.28 & 3.02 \\ 0 & 3.30 \end{bmatrix}$$

  and form reverse product

$$\boldsymbol{A}_1 = \boldsymbol{R}_1 \boldsymbol{Q}_1 = \begin{bmatrix} 7.83 & .906 \\ .906 & 3.17 \end{bmatrix}$$

- Off-diagonal entries are now smaller, and diagonal entries closer to eigenvalues, 8 and 3
- Process continues until matrix is within tolerance of being diagonal, and diagonal entries then closely approximate eigenvalues

QR Iteration with Shifts

- Convergence rate of QR iteration can be accelerated by incorporating *shifts*

$$\begin{aligned} \boldsymbol{Q}_k \boldsymbol{R}_k &= \boldsymbol{A}_{k-1} - \sigma_k \boldsymbol{I} \\ \boldsymbol{A}_k &= \boldsymbol{R}_k \boldsymbol{Q}_k + \sigma_k \boldsymbol{I} \end{aligned}$$

  where σ_k is rough approximation to eigenvalue

- Good shift can be determined by computing eigenvalues of 2×2 submatrix in lower right corner of matrix

Example: QR Iteration with Shifts

- Repeat previous example, but with shift of $\sigma_1 = 4$, which is lower right corner entry of matrix
- We compute QR factorization

$$\boldsymbol{A}_0 - \sigma_1 \boldsymbol{I} = \boldsymbol{Q}_1 \boldsymbol{R}_1 = \begin{bmatrix} .832 & .555 \\ .555 & -.832 \end{bmatrix} \begin{bmatrix} 3.61 & 1.66 \\ 0 & 1.11 \end{bmatrix}$$

  and form reverse product, adding back shift to obtain

$$\boldsymbol{A}_1 = \boldsymbol{R}_1 \boldsymbol{Q}_1 + \sigma_1 \boldsymbol{I} = \begin{bmatrix} 7.92 & .615 \\ .615 & 3.08 \end{bmatrix}$$

- After one iteration, off-diagonal entries smaller compared with unshifted algorithm, and eigenvalues closer approximations to eigenvalues

< interactive example >

Preliminary Reduction

- Efficiency of QR iteration can be enhanced by first transforming matrix as close to triangular form as possible before beginning iterations
- *Hessenberg matrix* is triangular except for one additional nonzero diagonal immediately adjacent to main diagonal
- Any matrix can be reduced to Hessenberg form in finite number of steps by orthogonal similarity transformation, for example using Householder transformations
- Symmetric Hessenberg matrix is tridiagonal
- Hessenberg or tridiagonal form is preserved during successive QR iterations

Preliminary Reduction, continued

Advantages of initial reduction to upper Hessenberg or tridiagonal form

- Work per QR iteration is reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ for general matrix or $\mathcal{O}(n)$ for symmetric matrix
- Fewer QR iterations are required because matrix nearly triangular (or diagonal) already
- If any zero entries on first subdiagonal, then matrix is block triangular and problem can be broken into two or more smaller subproblems

Preliminary Reduction, continued

- QR iteration is implemented in two-stages

$$\text{symmetric} \longrightarrow \text{tridiagonal} \longrightarrow \text{diagonal}$$

or

$$\text{general} \longrightarrow \text{Hessenberg} \longrightarrow \text{triangular}$$

- Preliminary reduction requires definite number of steps, whereas subsequent iterative stage continues until convergence
- In practice only modest number of iterations usually required, so much of work is in preliminary reduction
- Cost of accumulating eigenvectors, if needed, dominates total cost

Cost of QR Iteration

Approximate overall cost of preliminary reduction and QR iteration, counting both additions and multiplications

- Symmetric matrices
  - $\frac{4}{3}n^3$ for eigenvalues only
  - $9n^3$ for eigenvalues and eigenvectors
- General matrices
  - $10n^3$ for eigenvalues only
  - $25n^3$ for eigenvalues and eigenvectors

Krylov Subspace Methods

- Krylov subspace methods reduce matrix to Hessenberg (or tridiagonal) form using only matrix-vector multiplication
- For arbitrary starting vector $\boldsymbol{x}_0$, if

$$\boldsymbol{K}_k = \begin{bmatrix} \boldsymbol{x}_0 & \boldsymbol{A}\boldsymbol{x}_0 & \cdots & \boldsymbol{A}^{k-1}\boldsymbol{x}_0 \end{bmatrix}$$

then

$$\boldsymbol{K}_n^{-1}\boldsymbol{A}\boldsymbol{K}_n = \boldsymbol{C}_n$$

where $\boldsymbol{C}_n$ is upper Hessenberg (in fact, companion matrix)

- To obtain better conditioned basis for span$(\boldsymbol{K}_n)$, compute QR factorization

$$\boldsymbol{Q}_n\boldsymbol{R}_n = \boldsymbol{K}_n$$

so that

$$\boldsymbol{Q}_n^H\boldsymbol{A}\boldsymbol{Q}_n = \boldsymbol{R}_n\boldsymbol{C}_n\boldsymbol{R}_n^{-1} \equiv \boldsymbol{H}$$

with $\boldsymbol{H}$ upper Hessenberg

Krylov Subspace Methods

- Equating kth columns on each side of equation $\boldsymbol{A}\boldsymbol{Q}_n = \boldsymbol{Q}_n\boldsymbol{H}$ yields recurrence

$$\boldsymbol{A}\boldsymbol{q}_k = h_{1k}\boldsymbol{q}_1 + \cdots + h_{kk}\boldsymbol{q}_k + h_{k+1,k}\boldsymbol{q}_{k+1}$$

  relating $\boldsymbol{q}_{k+1}$ to preceding vectors $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k$

- Premultiplying by $\boldsymbol{q}_j^H$ and using orthonormality,

$$h_{jk} = \boldsymbol{q}_j^H \boldsymbol{A}\boldsymbol{q}_k, \quad j = 1, \ldots, k$$

- These relationships yield *Arnoldi iteration*, which produces unitary matrix $\boldsymbol{Q}_n$ and upper Hessenberg matrix $\boldsymbol{H}_n$ column by column using only matrix-vector multiplication by $\boldsymbol{A}$ and inner products of vectors

Arnoldi Iteration

$\boldsymbol{x}_0 =$ arbitrary nonzero starting vector
$\boldsymbol{q}_1 = \boldsymbol{x}_0 / \|\boldsymbol{x}_0\|_2$
for $k = 1, 2, \ldots$
  $\boldsymbol{u}_k = \boldsymbol{A}\boldsymbol{q}_k$
  for $j = 1$ **to** k
    $h_{jk} = \boldsymbol{q}_j^H \boldsymbol{u}_k$
    $\boldsymbol{u}_k = \boldsymbol{u}_k - h_{jk}\boldsymbol{q}_j$
  end
  $h_{k+1,k} = \|\boldsymbol{u}_k\|_2$
  if $h_{k+1,k} = 0$ **then** stop
  $\boldsymbol{q}_{k+1} = \boldsymbol{u}_k / h_{k+1,k}$
end

Arnoldi Iteration, continued

- If
$$\boldsymbol{Q}_k = \begin{bmatrix} \boldsymbol{q}_1 & \cdots & \boldsymbol{q}_k \end{bmatrix}$$
then
$$\boldsymbol{H}_k = \boldsymbol{Q}_k^H \boldsymbol{A} \boldsymbol{Q}_k$$
is upper Hessenberg matrix
- Eigenvalues of $\boldsymbol{H}_k$, called *Ritz values*, are approximate eigenvalues of $\boldsymbol{A}$, and *Ritz vectors* given by $\boldsymbol{Q}_k \boldsymbol{y}$, where $\boldsymbol{y}$ is eigenvector of $\boldsymbol{H}_k$, are corresponding approximate eigenvectors of $\boldsymbol{A}$
- Eigenvalues of $\boldsymbol{H}_k$ must be computed by another method, such as QR iteration, but this is easier problem if $k \ll n$

Arnoldi Iteration, continued

- Arnoldi iteration fairly expensive in work and storage because each new vector $\boldsymbol{q}_k$ must be orthogonalized against all previous columns of $\boldsymbol{Q}_k$, and all must be stored for that purpose.
- So Arnoldi process usually restarted periodically with carefully chosen starting vector
- Ritz values and vectors produced are often good approximations to eigenvalues and eigenvectors of $\boldsymbol{A}$ after relatively few iterations

Lanczos Iteration

- Work and storage costs drop dramatically if matrix is symmetric or Hermitian, since recurrence then has only three terms and $\boldsymbol{H}_k$ is tridiagonal (so usually denoted $\boldsymbol{T}_k$)

$\boldsymbol{q}_0 = \boldsymbol{0}$
$\beta_0 = 0$
$\boldsymbol{x}_0 =$ arbitrary nonzero starting vector
$\boldsymbol{q}_1 = \boldsymbol{x}_0 / \|\boldsymbol{x}_0\|_2$
for $k = 1, 2, \ldots$
  $\boldsymbol{u}_k = \boldsymbol{A}\boldsymbol{q}_k$
  $\alpha_k = \boldsymbol{q}_k^H \boldsymbol{u}_k$
  $\boldsymbol{u}_k = \boldsymbol{u}_k - \beta_{k-1}\boldsymbol{q}_{k-1} - \alpha_k \boldsymbol{q}_k$
  $\beta_k = \|\boldsymbol{u}_k\|_2$
  if $\beta_k = 0$ **then** stop
  $\boldsymbol{q}_{k+1} = \boldsymbol{u}_k / \beta_k$
end

Lanczos Iteration, continued

- α_k and β_k are diagonal and subdiagonal entries of symmetric tridiagonal matrix $\boldsymbol{T}_k$
- As with Arnoldi, Lanczos iteration does not produce eigenvalues and eigenvectors directly, but only tridiagonal matrix $\boldsymbol{T}_k$, whose eigenvalues and eigenvectors must be computed by another method to obtain Ritz values and vectors
- If $\beta_k = 0$, then algorithm appears to break down, but in that case invariant subspace has already been identified (i.e., Ritz values and vectors are already exact at that point)

Lanczos Iteration, continued

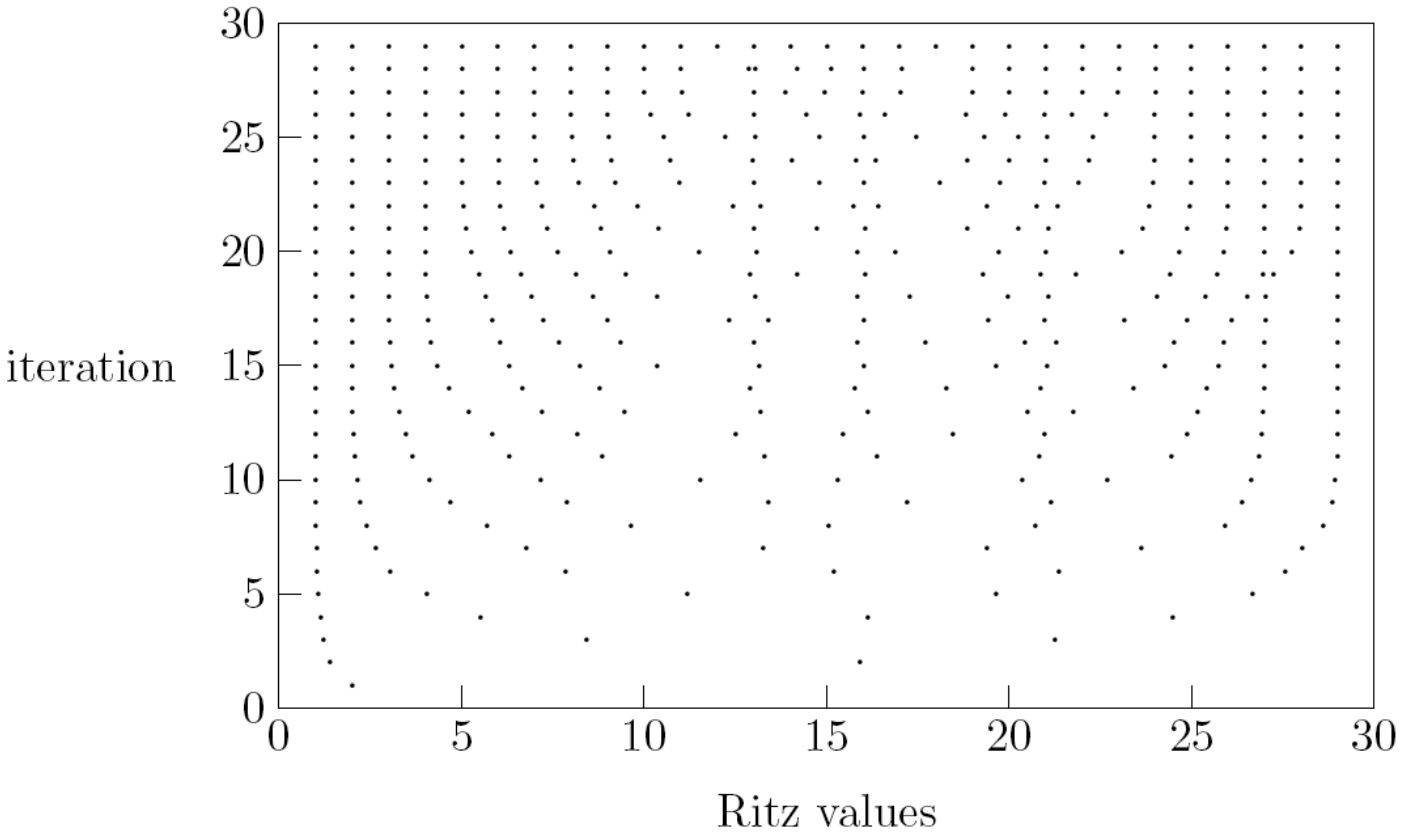
- In principle, if Lanczos algorithm were run until $k = n$, resulting tridiagonal matrix would be orthogonally similar to $\boldsymbol{A}$
- In practice, rounding error causes loss of orthogonality, invalidating this expectation
- Problem can be overcome by reorthogonalizing vectors as needed, but expense can be substantial
- Alternatively, can ignore problem, in which case algorithm still produces good eigenvalue approximations, but multiple copies of some eigenvalues may be generated

Krylov Subspace Methods, continued

- Great virtue of Arnoldi and Lanczos iterations is their ability to produce good approximations to extreme eigenvalues for $k \ll n$
- Moreover, they require only one matrix-vector multiplication by $\boldsymbol{A}$ per step and little auxiliary storage, so are ideally suited to large sparse matrices
- If eigenvalues are needed in middle of spectrum, say near σ, then algorithm can be applied to matrix $(\boldsymbol{A} - \sigma \boldsymbol{I})^{-1}$, assuming it is practical to solve systems of form $(\boldsymbol{A} - \sigma \boldsymbol{I})\boldsymbol{x} = \boldsymbol{y}$

Example: Lanczos Iteration

- For 29×29 symmetric matrix with eigenvalues $1, \ldots, 29$, behavior of Lanczos iteration is shown below



Jacobi Method

- One of oldest methods for computing eigenvalues is *Jacobi method*, which uses similarity transformation based on plane rotations
- Sequence of plane rotations chosen to annihilate symmetric pairs of matrix entries, eventually converging to diagonal form
- Choice of plane rotation slightly more complicated than in Givens method for QR factorization
- To annihilate given off-diagonal pair, choose c and s so that

$$\begin{aligned} \boldsymbol{J}^T \boldsymbol{A} \boldsymbol{J} &= \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a & b \\ b & d \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \\ &= \begin{bmatrix} c^2 a - 2csb + s^2 d & c^2 b + cs(a-d) - s^2 b \\ c^2 b + cs(a-d) - s^2 b & c^2 d + 2csb + s^2 a \end{bmatrix} \end{aligned}$$

is diagonal

Jacobi Method, continued

- Transformed matrix diagonal if

$$c^2 b + cs(a - d) - s^2 b = 0$$

- Dividing both sides by $c^2 b$, we obtain

$$1 + \frac{s}{c}\frac{(a-d)}{b} - \frac{s^2}{c^2} = 0$$

- Making substitution $t = s/c$, we get quadratic equation

$$1 + t\,\frac{(a-d)}{b} - t^2 = 0$$

  for tangent t of angle of rotation, from which we can recover $c = 1/(\sqrt{1+t^2})$ and $s = c \cdot t$

- Advantageous numerically to use root of smaller magnitude

Example: Plane Rotation

- Consider 2×2 matrix

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

- Quadratic equation for tangent reduces to $t^2 = 1$, so $t = \pm 1$
- Two roots of same magnitude, so we arbitrarily choose $t = -1$, which yields $c = 1/\sqrt{2}$ and $s = -1/\sqrt{2}$
- Resulting plane rotation $\boldsymbol{J}$ gives

$$\boldsymbol{J}^T \boldsymbol{A} \boldsymbol{J} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix}$$

Jacobi Method, continued

- Starting with symmetric matrix $\boldsymbol{A}_0 = \boldsymbol{A}$, each iteration has form
$$\boldsymbol{A}_{k+1} = \boldsymbol{J}_k^T \boldsymbol{A}_k \boldsymbol{J}_k$$
where $\boldsymbol{J}_k$ is plane rotation chosen to annihilate a symmetric pair of entries in $\boldsymbol{A}_k$
- Plane rotations repeatedly applied from both sides in systematic sweeps through matrix until off-diagonal mass of matrix is reduced to within some tolerance of zero
- Resulting diagonal matrix orthogonally similar to original matrix, so diagonal entries are eigenvalues, and eigenvectors are given by product of plane rotations

Jacobi Method, continued

- Jacobi method is reliable, simple to program, and capable of high accuracy, but converges rather slowly and difficult to generalize beyond symmetric matrices
- Except for small problems, more modern methods usually require 5 to 10 times less work than Jacobi
- One source of inefficiency is that previously annihilated entries can subsequently become nonzero again, thereby requiring repeated annihilation
- Newer methods such as QR iteration preserve zero entries introduced into matrix

Example: Jacobi Method

- Let $\boldsymbol{A}_0 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 2 & 1 \\ 2 & 1 & 1 \end{bmatrix}$

- First annihilate (1,3) and (3,1) entries using rotation

$$\boldsymbol{J}_0 = \begin{bmatrix} 0.707 & 0 & -0.707 \\ 0 & 1 & 0 \\ 0.707 & 0 & 0.707 \end{bmatrix}$$

to obtain

$$\boldsymbol{A}_1 = \boldsymbol{J}_0^T \boldsymbol{A}_0 \boldsymbol{J}_0 = \begin{bmatrix} 3 & 0.707 & 0 \\ 0.707 & 2 & 0.707 \\ 0 & 0.707 & -1 \end{bmatrix}$$

Example, continued

- Next annihilate (1,2) and (2,1) entries using rotation

$$\boldsymbol{J}_1 = \begin{bmatrix} 0.888 & -0.460 & 0 \\ 0.460 & 0.888 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

to obtain

$$\boldsymbol{A}_2 = \boldsymbol{J}_1^T \boldsymbol{A}_1 \boldsymbol{J}_1 = \begin{bmatrix} 3.366 & 0 & 0.325 \\ 0 & 1.634 & 0.628 \\ 0.325 & 0.628 & -1 \end{bmatrix}$$

Example, continued

- Next annihilate (2,3) and (3,2) entries using rotation

$$\boldsymbol{J}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.975 & -0.221 \\ 0 & 0.221 & 0.975 \end{bmatrix}$$

to obtain

$$\boldsymbol{A}_3 = \boldsymbol{J}_2^T \boldsymbol{A}_2 \boldsymbol{J}_2 = \begin{bmatrix} 3.366 & 0.072 & 0.317 \\ 0.072 & 1.776 & 0 \\ 0.317 & 0 & -1.142 \end{bmatrix}$$

Example, continued

- Beginning new sweep, again annihilate (1,3) and (3,1) entries using rotation

$$\boldsymbol{J}_3 = \begin{bmatrix} 0.998 & 0 & -0.070 \\ 0 & 1 & 0 \\ 0.070 & 0 & 0.998 \end{bmatrix}$$

to obtain

$$\boldsymbol{A}_4 = \boldsymbol{J}_3^T \boldsymbol{A}_3 \boldsymbol{J}_3 = \begin{bmatrix} 3.388 & 0.072 & 0 \\ 0.072 & 1.776 & -0.005 \\ 0 & -0.005 & -1.164 \end{bmatrix}$$

Example, continued

- Process continues until off-diagonal entries reduced to as small as desired
- Result is diagonal matrix orthogonally similar to original matrix, with the orthogonal similarity transformation given by product of plane rotations

< interactive example >

Bisection or Spectrum-Slicing

- For real symmetric matrix, can determine how many eigenvalues are less than given real number σ
- By systematically choosing various values for σ (*slicing spectrum* at σ) and monitoring resulting count, any eigenvalue can be isolated as accurately as desired
- For example, symmetric indefinite factorization $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$ makes *inertia* (numbers of positive, negative, and zero eigenvalues) of symmetric matrix $\boldsymbol{A}$ easy to determine
- By applying factorization to matrix $\boldsymbol{A} - \sigma\boldsymbol{I}$ for various values of σ, individual eigenvalues can be isolated as accurately as desired using interval bisection technique

Sturm Sequence

- Another spectrum-slicing method for computing individual eigenvalues is based on *Sturm sequence* property of symmetric matrices
- Let $\boldsymbol{A}$ be symmetric matrix and let $p_r(\sigma)$ denote determinant of leading principal minor of order r of $\boldsymbol{A} - \sigma \boldsymbol{I}$
- Then zeros of $p_r(\sigma)$ strictly separate those of $p_{r-1}(\sigma)$, and number of agreements in sign of successive members of sequence $p_r(\sigma)$, for $r = 1, \ldots, n$, equals number of eigenvalues of $\boldsymbol{A}$ strictly greater than σ
- Determinants $p_r(\sigma)$ are easy to compute if $\boldsymbol{A}$ is transformed to tridiagonal form before applying Sturm sequence technique

Divide-and-Conquer Method

- Express symmetric tridiagonal matrix $\boldsymbol{T}$ as

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{T}_1 & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{T}_2 \end{bmatrix} + \beta\, \boldsymbol{u}\boldsymbol{u}^T$$

- Can now compute eigenvalues and eigenvectors of smaller matrices $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$
- To relate these back to eigenvalues and eigenvectors of original matrix requires solution of *secular equation*, which can be done reliably and efficiently
- Applying this approach recursively yields *divide-and-conquer* algorithm for symmetric tridiagonal eigenproblems

Relatively Robust Representation

- With conventional methods, cost of computing eigenvalues of symmetric tridiagonal matrix is $\mathcal{O}(n^2)$, but if orthogonal eigenvectors are also computed, then cost rises to $\mathcal{O}(n^3)$
- Another possibility is to compute eigenvalues first at $\mathcal{O}(n^2)$ cost, and then compute corresponding eigenvectors separately using inverse iteration with computed eigenvalues as shifts
- Key to making this idea work is computing eigenvalues and corresponding eigenvectors to very high relative accuracy so that expensive explicit orthogonalization of eigenvectors is not needed
- RRR algorithm exploits this approach to produce eigenvalues *and* orthogonal eigenvectors at $\mathcal{O}(n^2)$ cost

Generalized Eigenvalue Problems

- *Generalized eigenvalue problem* has form

$$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{B}\boldsymbol{x}$$

  where $\boldsymbol{A}$ and $\boldsymbol{B}$ are given $n \times n$ matrices
- If either $\boldsymbol{A}$ or $\boldsymbol{B}$ is nonsingular, then generalized eigenvalue problem can be converted to standard eigenvalue problem, either

$$(\boldsymbol{B}^{-1}\boldsymbol{A})\boldsymbol{x} = \lambda \boldsymbol{x} \quad \text{or} \quad (\boldsymbol{A}^{-1}\boldsymbol{B})\boldsymbol{x} = (1/\lambda)\boldsymbol{x}$$

- This is not recommended, since it may cause
  - loss of accuracy due to rounding error
  - loss of symmetry if $\boldsymbol{A}$ and $\boldsymbol{B}$ are symmetric
- Better alternative for generalized eigenvalue problems is QZ algorithm

QZ Algorithm

- If $\boldsymbol{A}$ and $\boldsymbol{B}$ are triangular, then eigenvalues are given by $\lambda_i = a_{ii}/b_{ii}$, for $b_{ii} \neq 0$
- *QZ algorithm* reduces $\boldsymbol{A}$ and $\boldsymbol{B}$ simultaneously to upper triangular form by orthogonal transformations
- First, $\boldsymbol{B}$ is reduced to upper triangular form by orthogonal transformation from left, which is also applied to $\boldsymbol{A}$
- Next, transformed $\boldsymbol{A}$ is reduced to upper Hessenberg form by orthogonal transformation from left, while maintaining triangular form of $\boldsymbol{B}$, which requires additional transformations from right

QZ Algorithm, continued

- Finally, analogous to QR iteration, A is reduced to triangular form while still maintaining triangular form of B, which again requires transformations from both sides
- Eigenvalues can now be determined from mutually triangular form, and eigenvectors can be recovered from products of left and right transformations, denoted by Q and Z

Computing SVD

- *Singular values* of $\boldsymbol{A}$ are nonnegative square roots of eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$, and columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{A}^T\boldsymbol{A}$, respectively
- Algorithms for computing SVD work directly with $\boldsymbol{A}$, without forming $\boldsymbol{A}\boldsymbol{A}^T$ or $\boldsymbol{A}^T\boldsymbol{A}$, to avoid loss of information associated with forming these matrix products explicitly
- SVD is usually computed by variant of QR iteration, with $\boldsymbol{A}$ first reduced to bidiagonal form by orthogonal transformations, then remaining off-diagonal entries are annihilated iteratively
- SVD can also be computed by variant of Jacobi method, which can be useful on parallel computers or if matrix has special structure