Outline

1. Eigenvalue Problems
2. Existence, Uniqueness, and Conditioning
3. Computing Eigenvalues and Eigenvectors

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* $= \rho(\boldsymbol{A}) = \max\{|\lambda| : \lambda \in \lambda(\boldsymbol{A})\}$

Classic Eigenvalue Problem

- Consider the coupled pair of differential equations:

$$\begin{aligned} \frac{dv}{dt} &= 4v - 5w, \quad v = 8 \text{ at } t = 0, \\ \frac{dw}{dt} &= 2v - 3w, \quad w = 5 \text{ at } t = 0. \end{aligned}$$

- This is an ***initial-value problem***.

- With the coefficient matrix,

$$A = \begin{bmatrix} 4 & -5 \\ 2 & -3 \end{bmatrix},$$

  we can write this as,

$$\frac{d}{dt} \begin{pmatrix} v(t) \\ w(t) \end{pmatrix} = \begin{bmatrix} 4 & -5 \\ 2 & -3 \end{bmatrix} \begin{pmatrix} v(t) \\ w(t) \end{pmatrix}.$$

- Introducing the *vector unknown*, $\mathbf{u}(t) := [v(t)\ w(t)]^T$ with $\mathbf{u}(0) = [8\ 5]^T$, we can write the system in vector form,

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u}, \text{ with } \mathbf{u} = \mathbf{u}(0) \text{ at } t = 0.$$

- How do we find $\mathbf{u}(t)$?

- If we had a 1×1 matrix $A = a$, we would have a scalar equation:

$$\frac{du}{dt} \;=\; a\,u \;\text{ with } u = u(0) \text{ at } t = 0.$$

  The solution to this equation is a pure exponential:

$$u(t) \;=\; e^{at}\,u(0),$$

  which satisfies the initial contion because $e^0 = 1$.

- The derivative with respect to t is $ae^{at}u(0) = au$, so it satisfies the scalar initial value problem.

- The constant a is critical to how this system behaves.
  - If $a > 0$ then the solution grows in time.
  - If $a < 0$ then the solution decays.
  - If $a \in Im$ then the solution is oscillatory.
    (More on this later...)

- Coming back to our system, suppose we again look for solutions that are pure exponentials in time, e.g.,

$$\begin{aligned} v(t) &= e^{\lambda t} y \\ w(t) &= e^{\lambda t} z. \end{aligned}$$

- If this is to be a solution to our initial value problem, we require

$$\begin{aligned} \frac{dv}{dt} &= \lambda e^{\lambda t} y = 4e^{\lambda t} y - 5e^{\lambda t} z \\ \frac{dw}{dt} &= \lambda e^{\lambda t} z = 2e^{\lambda t} y - 3e^{\lambda t} z. \end{aligned}$$

- The $e^{\lambda t}$ cancels out from each side, leaving:

$$\begin{aligned} \lambda y &= 4y - 5z \\ \lambda z &= 2y - 3z, \end{aligned}$$

which is the eigenvalue problem.

$$\begin{bmatrix} 4 & -5 \\ 2 & -3 \end{bmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \begin{pmatrix} y \\ z \end{pmatrix}$$

- In vector form, $\mathbf{u}(t) = e^{\lambda t}\mathbf{x}$, yields

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u} \iff \lambda e^{\lambda t}\mathbf{x} = A(e^{\lambda t}\mathbf{x})$$

  which gives the eigenvalue problem in matrix form:

$$\lambda\mathbf{x} = A\mathbf{x} \quad \text{or}$$

$$A\mathbf{x} = \lambda\mathbf{x}.$$

- As in the scalar case, the solution behavior depends on whether λ has
  - positive real part $\longrightarrow$ a growing solution,
  - negative real part $\longrightarrow$ a decaying solution,
  - an imaginary part $\longrightarrow$ an oscillating solution.
- Note that here we have two unknowns: λ and $\mathbf{x}$.
- We refer to $(\lambda, \mathbf{x})$ as an eigenpair, with *eigenvalue* λ and *eigenvector* $\mathbf{x}$.

Solving the Eigenvalue Problem

- The eigenpair satisfies

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

  which is to say,

  – $\mathbf{x}$ is in the null-space of $A - \lambda I$

  – λ is chosen so that $A - \lambda I$ has a null-space.

- We thus seek λ such that $A - \lambda I$ is singular.

- Singularity implies $det(A - \lambda I) = 0$.

- For our example:

$$0 = \begin{vmatrix} 4-\lambda & -5 \\ 2 & -3-\lambda \end{vmatrix} = (4-\lambda)(-3-\lambda) - (-5)(2),$$

  or

$$\lambda^2 - \lambda - 2 = 0,$$

  which has roots $\lambda = -1$ or $\lambda = 2$.

Finding the Eigenvectors

- For the case $\lambda = \lambda_1 = -1$, $(A - \lambda_1 I)\mathbf{x}_1$ satisfies,

$$\begin{bmatrix} 5 & -5 \\ 2 & -2 \end{bmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

  which gives us the eigenvector $\mathbf{x}_1$

$$\mathbf{x}_1 = \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

- **Note that any nonzero multiple of $\mathbf{x}_1$ is also an eigenvector.**
- Thus, $\mathbf{x}_1$ defines a *subspace* that is invariant under multiplication by A.

- For the case $\lambda = \lambda_2 = 2$, $(A - \lambda_2 I)\mathbf{x}_2$ satisfies,

$$\begin{bmatrix} 2 & -5 \\ 2 & -5 \end{bmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

  which gives us the second eigenvector as any multiple of

$$\mathbf{x}_2 = \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} 5 \\ 2 \end{pmatrix}.$$

Return to Model Problem

- Note that our model problem $\frac{d\mathbf{u}}{dt} = A\mathbf{u}$, is *linear* in the unknown $\mathbf{u}$.
- Thus, if we have two solutions $\mathbf{u}_1(t)$ and $\mathbf{u}_2(t)$ satisfying the differential equation, their sum $\mathbf{u} := \mathbf{u}_1 + \mathbf{u}_2$ also satisfies the equation:

$$\begin{aligned} & \frac{d\mathbf{u}_1}{dt} = A\mathbf{u}_1 \\ + \quad & \frac{d\mathbf{u}_2}{dt} = A\mathbf{u}_2 \\ \hline & \frac{d}{dt}(\mathbf{u}_1 + \mathbf{u}_2) = A(\mathbf{u}_1 + \mathbf{u}_2) \\ & \frac{d\mathbf{u}}{dt} = A\mathbf{u} \end{aligned}$$

- Take $\mathbf{u}_1 = c_1 e^{\lambda_1 t}\mathbf{x}_1$:

$$\begin{aligned} \frac{d\mathbf{u}_1}{dt} &= c_1\lambda_1 e^{\lambda_1 t}\mathbf{x}_1 \\ A\mathbf{u}_1 &= A\left(c_1 e^{\lambda_1 t}\mathbf{x}_1\right) \\ &= c_1 e^{\lambda_1 t} A\mathbf{x}_1 \\ &= c_1 e^{\lambda_1 t}\lambda_1\mathbf{x}_1 \\ &= \frac{d\mathbf{u}_1}{dt}. \end{aligned}$$

- Similarly, for $\mathbf{u}_2 = c_2 e^{\lambda_2 t}\mathbf{x}_2$: $\dfrac{d\mathbf{u}_2}{dt} = A\mathbf{u}_2.$

- Thus,

$$\frac{d\mathbf{u}}{dt} = \frac{d}{dt}\left(\mathbf{u}_1 + \mathbf{u}_2\right) = A\left(\mathbf{u}_1 + \mathbf{u}_2\right)$$

$$\mathbf{u} = c_1 e^{\lambda_1 t}\mathbf{x}_1 + c_2 e^{\lambda_2 t}\mathbf{x}_2.$$

- The only remaining part is to find the coefficients c_1 and c_2 such that $\mathbf{u} = \mathbf{u}(0)$ at time $t = 0$.

- This initial condition yields a 2×2 system,

$$\begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 8 \\ 5 \end{pmatrix}.$$

- Solving for c_1 and c_2 via Gaussian elimination:

$$\begin{bmatrix} 1 & 5 \\ 1 & 2 \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 8 \\ 5 \end{pmatrix}$$

$$\begin{bmatrix} 1 & 5 \\ 0 & -3 \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 8 \\ -3 \end{pmatrix}$$

$$\begin{aligned} c_2 &= 1 \\ c_1 &= 8 - 5c_1 = 3. \end{aligned}$$

- So, our solution is $\mathbf{u}(t) = \mathbf{x}_1 c_1 e^{\lambda_1 t} + \mathbf{x}_2 c_2 e^{\lambda_2 t}$

$$= \begin{pmatrix} 1 \\ 1 \end{pmatrix} 3e^{-t} + \begin{pmatrix} 5 \\ 2 \end{pmatrix} e^{2t}.$$

- Clearly, after a long time, the solution is going to look like a multiple of $\mathbf{x}_2 = [5\ 2]^T$ because the component of the solution parallel to $\mathbf{x}_1$ will decay.
- (More precisely, the component parallel to $\mathbf{x}_1$ will not grow as fast as the component parallel to $\mathbf{x}_2$.)

Summary

- Model problem, $\mathbf{u} \in \mathcal{R}^n$,

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u}, \quad \mathbf{u} = \mathbf{u}(0) \text{ at time } t = 0.$$

- Assuming A has n *linearly independent* eigenvectors, can express

$$\mathbf{u}(t) = \sum_{j=1}^{n} \mathbf{x}_j c_j e^{\lambda_j t}.$$

- Coefficients c_j determined by initial condition:

$$X\mathbf{c} = \sum_{j=1}^{n} \mathbf{x}_j c_j = \mathbf{u}(0).$$

- Eigenpairs $(\lambda_j, \mathbf{x}_j)$ satisfy

$$A\mathbf{x}_j = \lambda_j \mathbf{x}_j.$$

Growing / Decaying Modes

- Our model problem,

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u} \longrightarrow \mathbf{u}(t) = \mathbf{x}_1 c_1 e^{\lambda_1 t} + \mathbf{x}_2 c_2 e^{\lambda_2 t}$$

  leads to growth/decay of components.

- Also get growth/decay through matrix-vector products.
- Consider $\mathbf{u} = c_1\mathbf{x}_1 + c_2\mathbf{x}_2$.

$$\begin{aligned} A\mathbf{u} &= c_1 A\mathbf{x}_1 + c_2 A\mathbf{x}_2 \\ &= c_1\lambda_1\mathbf{x}_1 + c_2\lambda_2\mathbf{x}_2 \end{aligned}$$

$$\begin{aligned} A^k\mathbf{u} &= c_1\lambda_1^k\mathbf{x}_1 + c_2\lambda_2^k\mathbf{x}_2 \\ &= \lambda_2^k\left[c_1\left(\frac{\lambda_1}{\lambda_2}\right)^k\mathbf{x}_1 + c_2\mathbf{x}_2\right]. \end{aligned}$$

$$\lim_{k\longrightarrow\infty} A^k\mathbf{u} = \lambda_2^k\left[c_1\cdot 0\cdot\mathbf{x}_1 + c_2\mathbf{x}_2\right] = c_2\lambda_2^k\mathbf{x}_2.$$

- So, repeated matrix-vector products lead to emergence of eigenvector associated with the eigenvalue λ that has largest modulus.
- This is the main idea behind the ***power method***, which is a common way to find the eigenvector associated with $\max|\lambda|$.

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

Companion Matrix, n=3

- Look at determinant of (C – I λ): | C- I λ | = 0

$$C := \begin{pmatrix} 0 & 0 & -c_0 \\ 1 & 0 & -c_1 \\ 0 & 1 & -c_2 \end{pmatrix}$$

$$\text{Eigenvalues: } |C - I\lambda| = \begin{vmatrix} -\lambda & 0 & -c_0 \\ 1 & -\lambda & -c_1 \\ 0 & 1 & (-c_2 - \lambda) \end{vmatrix}$$

$$= -c_0 \begin{vmatrix} 1 & -\lambda \\ 0 & 1 \end{vmatrix} + c_1 \begin{vmatrix} -\lambda & \\ 0 & 1 \end{vmatrix} - (c_2 + \lambda) \begin{vmatrix} -\lambda & 0 \\ 1 & -\lambda \end{vmatrix}$$

$$= -c_0 - c_1\lambda - c_2\lambda^2 - \lambda^3 = 0$$

$$\Longrightarrow p(\lambda) = c_0 + c_1\lambda + c_2\lambda^2 + \lambda^3 = 0$$

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

Algebraic Multiplicity

- Eigenvalue of multiplicity 1 is *simple*
- *Defective* matrix has eigenvalue of multiplicity $k > 1$ with fewer than k linearly independent corresponding eigenvectors

Geometric Multiplicity

- 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

Note: Every matrix is ϵ away from being diagonalizable.

Diagonalization

- The real merit of eigenvalue decomposition is that it simplifies powers of a matrix.
- Consider

$$
\begin{aligned}
X^{-1}AX &= D,\ \textit{diagonal} \\
AX &= XD \\
A &= XDX^{-1} \\
A^2 &= \left(XDX^{-1}\right)\left(XDX^{-1}\right) \\
&= XD^2X^{-1} \\
A^k &= \left(XDX^{-1}\right)\left(XDX^{-1}\right)\cdots\left(XDX^{-1}\right) \\
&= XD^kX^{-1} \\
&= X\begin{bmatrix} \lambda_1^k & & & \\ & \lambda_2^k & & \\ & & \ddots & \\ & & & \lambda_n^k \end{bmatrix}X^{-1}
\end{aligned}
$$

- High powers of A tend to be dominated by largest eigenpair $(\lambda_1, \underline{x}_1)$, assuming $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$.

Matrix Powers Example

- Consider our 1D finite difference example introduced earlier.

$$\frac{d^2u}{dx^2} \;=\; f(x) \;\longrightarrow\; -\frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} \;\approx\; f(x_i).$$

  where $u(0) = u(1) = 0$ and $\Delta x = 1/(n+1)$.

- In matrix form,

$$A\mathbf{u} = \frac{1}{\Delta x^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_m \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_m \end{pmatrix}$$

- Eigenvectors and eigenvalues have closed-form expression:

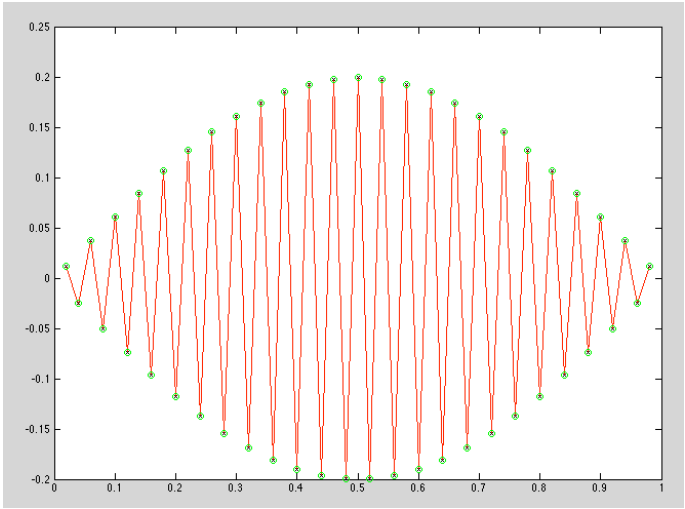
$$(\mathbf{z}_k)_i \;=\; \sin k\pi x_i \;=\; \sin k\pi i\Delta x \qquad \lambda_k \;=\; \frac{2}{\Delta x^2}\left(1 - \cos k\pi\Delta x\right)$$

- Eigenvalues are in the interval $\sim\; [\pi^2,\; 4(n+1)^2]$.

Matlab Example: heat_demo.m

- Repeatedly applying A to a random input vector reveals the eigenvalue of maximum modulus.
- This idea leads to one of the most common (but not most efficient) ways of finding an eigenvalue/vector pair, called the power method.

```
h = 1/(n+1);
e = ones(n,1);
A = spdiags([-e 2*e -e],-1:1, n,n)/(h*h);

x=1:n; x=h*x';

z=rand(n,1); hold off;
for k=1:2000;
    z=A*z; z=z/norm(z);
    if mod(k,100)==0; plot(x,z,'r-'); pause(.1); end;
end;
```



Diagonalization

- Note that if we define $A^0 = I$, we have any polynomial of A defined as

$$p_k(A)\underline{x} = X \begin{bmatrix} p_k(\lambda_1) & & & \\ & p_k(\lambda_2) & & \\ & & \ddots & \\ & & & p_k(\lambda_n) \end{bmatrix} X^{-1}\underline{x}.$$

- We can further extend this to other functions,

$$f(A)\underline{x} = X \begin{bmatrix} f(\lambda_1) & & & \\ & f(\lambda_2) & & \\ & & \ddots & \\ & & & f(\lambda_n) \end{bmatrix} X^{-1}\underline{x}.$$

- For example, the solution to $f(A)\underline{x} = \underline{b}$ is would be

$$\underline{x} = X\left[f(D)\right]^{-1} X^{-1}\underline{b}.$$

- The diagonalization concept is very powerful because it transforms *systems* of equations into scalar equations.

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

In theory…

When might invariance fail??

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	$\boldsymbol{A}^H\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^H$

Skew symmetric: $\mathbf{A} = -\mathbf{A}^T$

Upper Hessenberg (from last lecture…)

- A is upper Hessenberg – *A is upper triangular with one additional nonzero diagonal below the main one:* $A_{ij} = 0$ *if* $i > j+1$

0.1967	0.2973	0.0899	0.3381	0.5261	0.3965	0.1279
0.0934	0.0620	0.0809	0.2940	0.7297	0.0616	0.5495
0	0.2982	0.7772	0.7463	0.7073	0.7802	0.4852
0	0	0.9051	0.0103	0.7814	0.3376	0.8905
0	0	0	0.0484	0.2880	0.6079	0.7990
0	0	0	0	0.6925	0.7413	0.7343
0	0	0	0	0	0.1048	0.0513

- Requires only n Givens rotations, instead of $O(n^2)$, to effect QR factorization.

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: $\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$ Skew-Symmetric: $\begin{bmatrix} 0 & -2 \\ 2 & 0 \end{bmatrix} = -\begin{bmatrix} 0 & -2 \\ 2 & 0 \end{bmatrix}^T$
- 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: $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$

← *"canonical non-normal matrix"*
Defective – has only one eigenvector.

Normal Matrices

Normal matrices have orthogonal eigenvectors, so $\underline{x}_i^H \underline{x}_j = \delta_{ij}$

Beware!

- If A is normal, it *has* orthogonal eigenvectors.
- That does not mean that all eigensolvers will *return* orthogonal eigenvectors.
- In particular, if two or more eigenvectors share the same eigenvalue, then they needn't be orthogonal to each other.
- You probably need to orthogonalize them yourself.

Normal ma

- symme
- skew-symmetric $(A = -A^T)$
- unitary $(U^H U = I)$
- circulant (periodic+Toeplitz)
- others ...

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix}$$

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? *Sparsity*

Sparsity

- Sparsity, either direct or implied, is a big driver in choice of eigenvalue solvers.
- Typically, only O(n) entries in entire matrix, where n ~ 10^9—10^{18} might be anticipated.
- Examples include Big Data (e.g., google page rank) and physics simulations (fluid, heat transfer, electromagnetics, fusion, etc.).
- Usually, need only a few eigenvectors / eigenvalues, k << n.
- Often, there are special properties of A that make it difficult to create A. Instead, work strictly with matrix-vector products

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

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

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

  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

Simple non-diagonalizable example, 2×2 Jordan block:

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \lambda \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\begin{vmatrix} 1-\lambda & 1 \\ 0 & 1-\lambda \end{vmatrix} = (1-\lambda)^2 = 0$$

Only one eigenvector: $\underline{x} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

3×3 **Non-Diagonalizable Example**

$$A = \begin{bmatrix} 2 & & \\ & 2 & \\ & & 2 \end{bmatrix}, \qquad B = \begin{bmatrix} 2 & 1 & \\ & 2 & 1 \\ & & 2 \end{bmatrix}.$$

- Characteristic polynomial is $(\lambda - 2)^3$ for both A and B.
- Algebraic multiplicity is 3.
- For A, three eigenvectors. Say, $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$.
- For B, only one eigenvector ($\alpha\mathbf{e}_1$), so geometric multiplicity of B is 1.

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

Eigenvectors / Eigenvalues of Upper Triangular Matrix

Suppose A is upper triangular

$$A = \begin{bmatrix} A_{11} & \underline{u} & U_{13} \\ 0 & \lambda & \underline{v}^T \\ O & 0 & A_{33} \end{bmatrix}$$

Then

$$0 = (A - \lambda I)\underline{x} = \begin{bmatrix} U_{11} & \underline{u} & U_{13} \\ 0 & 0 & \underline{v}^T \\ O & 0 & U_{33} \end{bmatrix} \begin{pmatrix} \underline{y} \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} U_{11}\underline{y} - \underline{u} \\ 0 \\ 0 \end{pmatrix}$$

$$(A - \lambda I) \qquad \underline{x} \qquad 0$$

Because U_{11} is nonsingular, can solve $U_{11}\underline{y} = \underline{u}$ to find eigenvector $\underline{x}$.

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

Eigenvalue-Revealing Factorizations

- Diagonalization: $A = X\Lambda X^{-1}$ if A is nondefective.
- Unitary diagonalization: $A = Q\Lambda Q^*$ if A is normal.
- Unitary triangularization: $A = QTQ^*$ always exists. (T upper triangular.)

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)	***Always exists***
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

Similarity Transformations

- Given

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

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

- If A is normal ($AA^H = A^H A$),

$$A = Q \Lambda Q^H$$

  B is diagonal, T is unitary ($T^{-1} = T^H$).

- If A is symmetric real,

$$A = Q \Lambda Q^T$$

  B is diagonal, T is orthogonal ($T^{-1} = T^T$).

- If B is diagonal, T is the matrix of eigenvectors.

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 $\boldsymbol{v}_1$

2 x 2 Example

$$A = \begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & \\ & 2 \end{bmatrix} \qquad X = \begin{bmatrix} -1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

Example: Power Iteration

- Ratio of values of given component of x_k from one iteration to next converges to dominant eigenvalue λ_1
- For example, if $A = \begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix}$ and $x_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, we obtain

$$x_k = Ax_{k-1}$$

k	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, $\|\boldsymbol{y}_k\|_\infty \rightarrow |\lambda_1|$, and $\boldsymbol{x}_k \rightarrow \boldsymbol{v}_1 / \|\boldsymbol{v}_1\|_\infty$

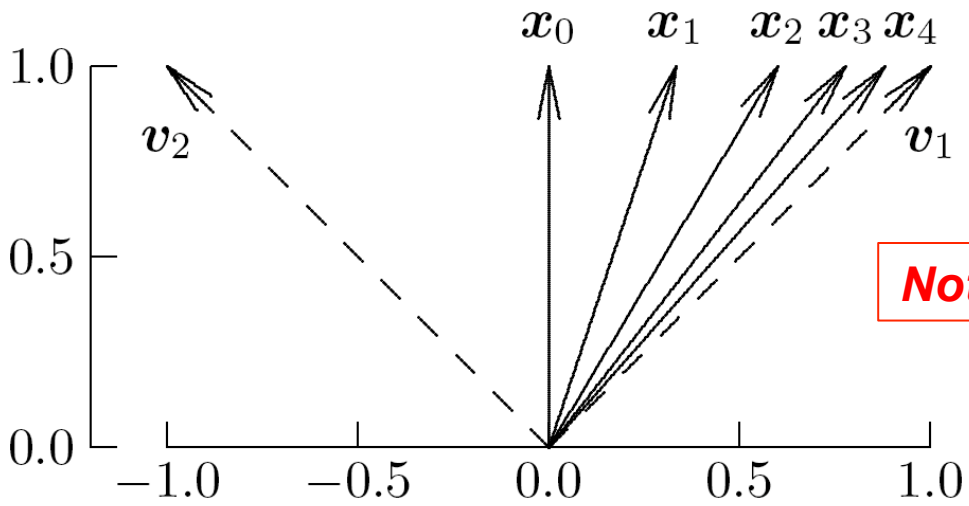
Example: Normalized Power Iteration

- 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

Geometric Interpretation

- Behavior of power iteration depicted geometrically



Note:* ∞*-norm = 1.

- 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

Convergence Rate of Power Iteration

Convergence rate of power iteration depends on relative separation of λ_1 and λ_2.

Assuming $c_1 \neq 0$ and $|\lambda_1| > |\lambda_j|$, $j > 1$, we have

$$
\begin{aligned}
A^k \underline{x} &= \sum_{j=1}^{n} \underline{x}_j \, \lambda_j^k \, c_j \\
&= \lambda_1^k \, c_1 \left[\underline{x}_1 + \sum_{j=2}^{n} \underline{x}_j \, \frac{\lambda_j^k}{\lambda_1^k} \, \frac{c_j}{c_1} \right] \\
&\sim \underline{x}_1 \, \lambda_1^k \, c_1 \text{ as } k \longrightarrow \infty \\
&\sim \lambda_1^k \, c_1 \left[\underline{x}_1 + \underline{x}_2 \left(\frac{\lambda_2}{\lambda_1} \right)^k \frac{c_2}{c_1} \right]
\end{aligned}
$$

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

Note: in general, need $\max_j |\lambda_j - \sigma| / |\lambda_1 - \sigma| < |\lambda_2| / |\lambda_1|$.

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

Idea: pick shifts at each iteration as estimate of spectrum emerges.

Power Iteration with Shift

$$(A - \sigma I)\underline{x} \;=\; \lambda\underline{x} - \sigma\underline{x} \;=\; (\lambda - \sigma)\underline{x} \;=\; \mu\underline{x}$$

If $\lambda_k \in \{1\ .9\ \ldots\ .1\}$, then

$$\frac{\lambda_2}{\lambda_1} = 0.9$$

If $\sigma = 0.4$, then $\mu_k \in \{.5\ .4\ \ldots\ -.4\}$ and

$$\frac{\mu_2}{\mu_1} = 0.8,$$

so about twice the convergence rate.

Shifted power iteration, however, is somewhat limited.

The real power derives from *inverse* power iterations with shifts.

Inverse Iteration

Very Important!

- 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)

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

- Power Iteration: $\mathbf{x} = A^k \mathbf{x} \longrightarrow c\mathbf{x}_1$

- Normalized Power Iteration:
$$\left.\begin{array}{l} \mathbf{y} = A\mathbf{x} \\ \mathbf{x} = \mathbf{y}/||\mathbf{y}|| \end{array}\right\} \begin{array}{l} ||\mathbf{y}|| \longrightarrow |\lambda_1| \\ \mathbf{x} \longrightarrow \mathbf{x}_1 \end{array}$$

- Inverse Iteration:
$$\left.\begin{array}{l} \mathbf{y} = A^{-1}\mathbf{x} \\ \mathbf{x} = \mathbf{y}/||\mathbf{y}|| \end{array}\right\} \begin{array}{l} ||\mathbf{y}|| \longrightarrow |\lambda_n|^{-1} \\ \mathbf{x} \longrightarrow \mathbf{x}_n \end{array}$$

- Inverse Iteration with shift:
$$\left.\begin{array}{l} M = A - \sigma I \\ \mathbf{y} = M^{-1}\mathbf{x} \\ \mathbf{x} = \mathbf{y}/||\mathbf{y}|| \end{array}\right\} \begin{array}{l} ||\mathbf{y}|| \longrightarrow |\mu_k| = \max |\lambda_k - \sigma|^{-1} \\ \mathbf{x} \longrightarrow \mathbf{x}_k \end{array}$$

Inverse iteration with shift can be arbitrarily fast since separation ratio can be 0.

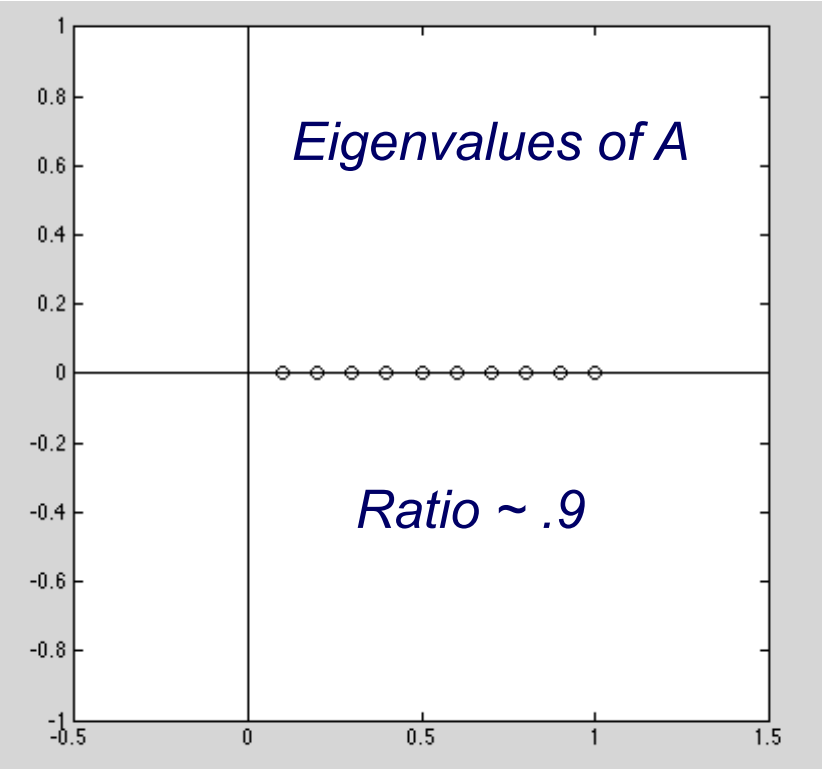
- Inverse Iteration:

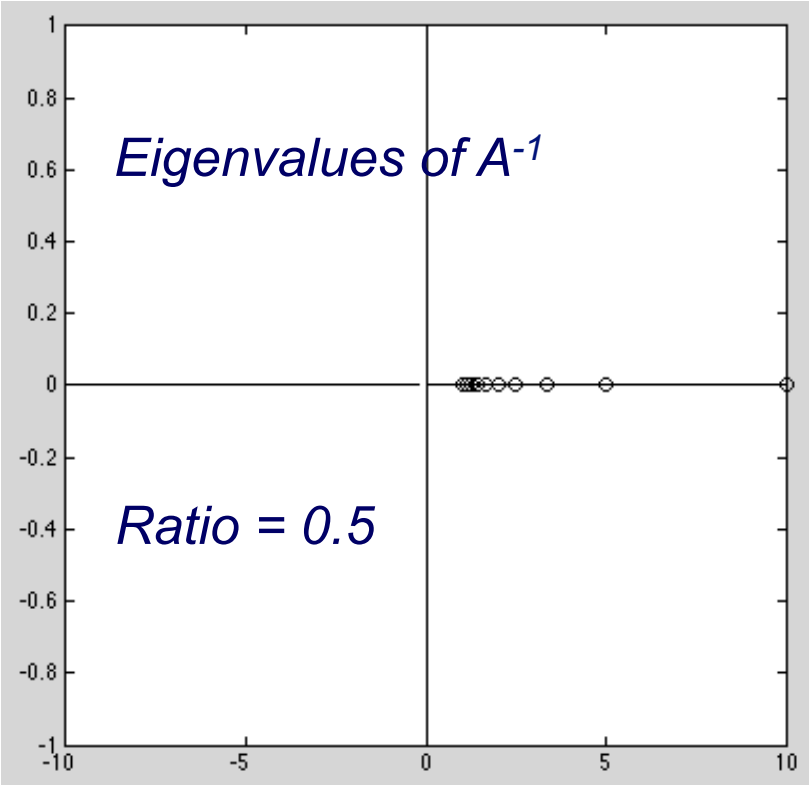
$$\begin{aligned}
\mathbf{x}^k &= A^{-k}\mathbf{x}^0 \\
&= \sum_{j=1}^{n} \mathbf{x}_j \left(\frac{1}{\lambda_j}\right)^k c_j \\
&= \left(\frac{1}{\lambda_n}\right)^k c_n \left[\mathbf{x}_n + \sum_{j=1}^{n-1} \mathbf{x}_j \left(\frac{\lambda_n}{\lambda_j}\right)^k c_j\right].
\end{aligned}$$

eig_inv_demo.m

Inverse Iteration Illustration

- With shift and invert, can get significant ratios of dominant eigenvalue





- Inverse Iteration with Shift: Let

$$M := A - \sigma I$$

$$\mu_j := \lambda_j - \sigma, \text{ and}$$

$$l \text{ such that } |\sigma - \mu_l| < |\sigma - \mu_j|,\ j \neq l.$$

Then,

$$\begin{aligned}
\mathbf{x}^k &= M^{-k}\mathbf{x}^0 \\
&= \sum_{j=1}^{n} \mathbf{x}_j \left(\frac{1}{\mu_j}\right)^k c_j \\
&= \left(\frac{1}{\mu_l}\right)^k c_l \left[\mathbf{x}_l + \sum_{j \neq l} \mathbf{x}_j \left(\frac{\mu_l}{\mu_j}\right)^k c_j\right].
\end{aligned}$$

- Using current approximation to λ_l can select $|\sigma - \lambda_l| = |\mu_l|$ to be small. (Cannot do the same with shifted power iteration.)
- Blow-up is contained by normalizing after each iteration.

eig_shift_invert.m

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

Rayleigh Quotient Convergence

- Assume A is a symmetric matrix and $\mathbf{x}$ is close to an eigenvector,

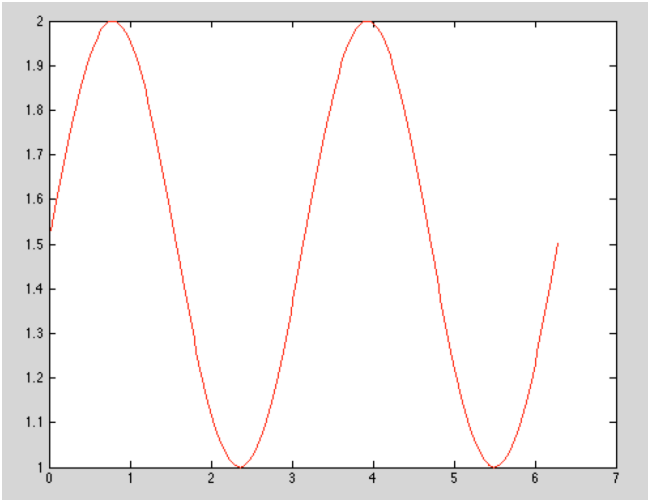
$$\mathbf{x} \;=\; c_1\mathbf{x}_1 \;+\; \sum_{j=2}^{n} c_j\,\mathbf{x}_j, \;\; |c_j| \;=\; \epsilon \text{ for } j \;\geq\; 2$$

- If $||\mathbf{x}|| = 1$, then the Rayleigh quotient

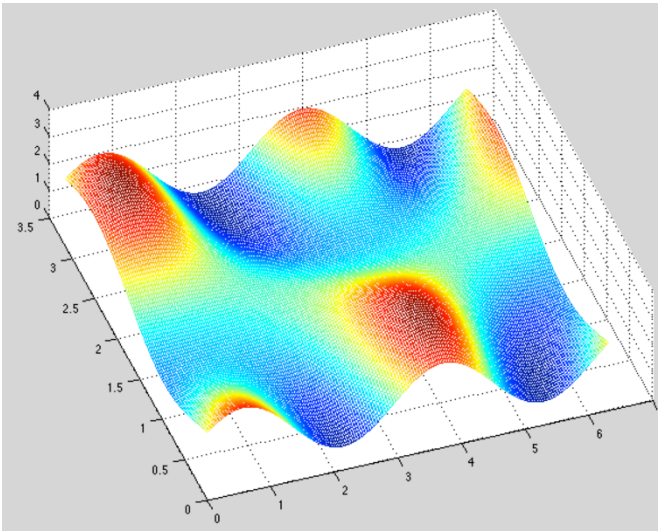
$$\begin{aligned} r(\mathbf{x}) \;&=\; \frac{\mathbf{x}^T A\;\mathbf{x}}{\mathbf{x}^T\mathbf{x}} \;=\; \mathbf{x}^T A\;\mathbf{x} \\ &=\; \sum_{i=1}^{n}\sum_{j=1}^{n} c_i\,c_j\,\lambda_i \mathbf{x}_i^T\mathbf{x}_j \\ &=\; \sum_{i=1}^{n} c_i^2\,\lambda_i \\ &=\; \left(1 - n\epsilon^2\right)\lambda_1 \;+\; \epsilon^2 \sum_{i=2}^{n}\lambda_i \end{aligned}$$

- Rayleigh quotient is stationary at eigenvectors.
- It takes on min/max when j=n or 1.

```
% Plot Rayleigh Quotient on S1 for 2 x 2 matrix

V=[ 1   1 ; -1   1 ];
D=[ 1   0 ;  0   2 ];
for k=1:2; V(:,k)=V(:,k)/norm(V(:,k)); end;

A=V*D*inv(V);

m=200; dt=2*pi/m;

for k=1:m; t=k*dt;
  x=[ cos(t) ; sin(t) ];
  z=A*x;
  rq(k)=z'*x;
  th(k)=t;
end;

plot(th,rq,'r-');
```




```
% Plot Rayleigh Quotient on S2 for 3 x 3 matrix

V=[ 1   1   1 ; -1   1   1; 1   1   0];
D=[ 1   0   0 ;  0   2   0; 0 0 3];
A=V*D*inv(V);

m=200; dt=2*pi/m; t=0:dt:2*pi;    %% theta
n=100; dp=pi/n;   p=0:dp:pi;      %% phi
[T,P]=ndgrid(t,p);

r=cos(P); z=sin(P); x=r.*cos(T); y=r.*sin(T);

rq=0*x;
for i=1:m+1; for j=1:n+1;
   v = [ x(i,j); y(i,j); z(i,j)];  w=A*v;
   rq(i,j)= v'*w;
end; end;

mesh(T,P,rq)
```



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*

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

$$(\boldsymbol{A} - \sigma_k \boldsymbol{I}) \boldsymbol{y}_{k+1} = \boldsymbol{x}_k \qquad \textit{Solve a system!}$$

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

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

Note:

- You don't need to refactor A - σ I for each iteration.
- You can take several iterations, then refactor.

Example: Rayleigh Quotient Iteration

- Using same matrix as previous examples and randomly chosen starting vector x_0, Rayleigh quotient iteration converges in two iterations

k	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 – Finding Second Eigenpair

- Choose H to be elementary Householder matrix such that $H\mathbf{x}_1 = \mathbf{e}_1$.
- Consider

$$\begin{aligned} A\mathbf{x}_1 &= \lambda_1 \mathbf{x}_1 \\ AH^{-1}H\mathbf{x}_1 &= \lambda_1 H^{-1}H\mathbf{x}_1 \\ AH^{-1}\mathbf{e}_1 &= \lambda_1 H^{-1}\mathbf{e}_1 \\ HAH^{-1}\mathbf{e}_1 &= \lambda_1 \mathbf{e}_1 \qquad * \end{aligned}$$

$$\begin{aligned} A_2 := HAH^{-1} &= HAH^{-1} I \\ &= HAH^{-1} [\, \mathbf{e}_1 \; \mathbf{e}_2 \; \ldots \; \mathbf{e}_n \,] \\ &= \begin{bmatrix} \lambda_1 & \mathbf{b}^T \\ 0 & \\ 0 & B \\ 0 & \end{bmatrix} \end{aligned}$$

- Apply method of choice to B to find λ_2.

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

Matrix-Free Deflation

Deflation via Projection.

for $k = 1, 2, \ldots$

$$\begin{aligned} \mathbf{y} &= A\mathbf{x} \\ \mathbf{y} &= \mathbf{y} - \mathbf{x}_1 \frac{\mathbf{x}_1^T \mathbf{y}}{\mathbf{x}_1^T \mathbf{x}_1} = \mathbf{y} - \mathbf{x}_1 \mathbf{x}_1^T \mathbf{y} \\ \mathbf{x} &= \mathbf{y}/||\mathbf{y}||. \end{aligned}$$

- Guarantees that $\mathbf{x}$ is devoid of any component of $\mathbf{x}_1$ prior to start of each application of power method.
- Do not require knowledge of A.
- Only need a routine that provides $\mathbf{y} \longleftarrow A\mathbf{x}$.
- Convenient for large sparse matrices when trying to avoid computing known eigenpairs.

Deflation on the Fly – *Subspace Iteration*

Can effect deflation on the fly — *Subspace Iteration*.

- Take two independent vectors $Y = (\mathbf{y}_1\ \mathbf{y}_2)$.

$$\text{for } k = 1,\ 2,\ldots$$

$$\begin{aligned} Z &= AY \\ \mathbf{y}_1 &= \mathbf{z}_1/||\mathbf{z}_1||. \\ \mathbf{y}_2 &= \mathbf{z}_2 - \mathbf{y}_1\mathbf{y}_1^T\mathbf{z}_2 \\ \mathbf{y}_2 &= \mathbf{y}_2/||\mathbf{y}_2|| \end{aligned}$$

- $(\mathbf{y}_1\ \mathbf{y}_2)$ converge to the first two eigenvectors $(\mathbf{x}_1\ \mathbf{x}_2)$.

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

Subspace Iteration Variants

Let $X_0 \in \mathbb{R}^{n \times p}$ be a matrix of rank p.

- Alg. 1:

  for $k = 1, 2, \ldots$
  $X_k = A X_{k-1}$
  end

- Alg. 2:

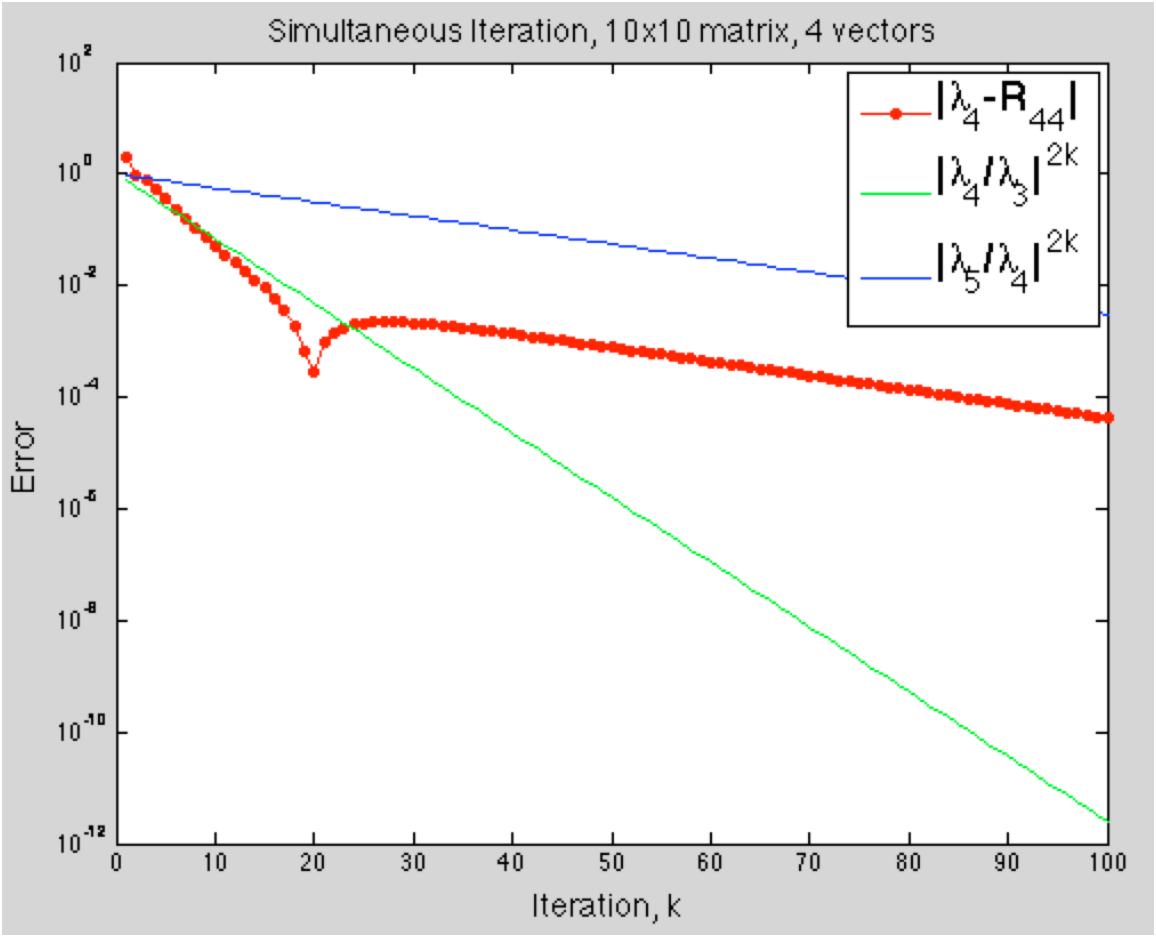
  for $k = 1, 2, \ldots$
  $Q R = X$ $\quad$ $Q \in \mathbb{R}^{n \times p}$, orthogonal
  $X = A Q$
  end

Matlab Demo: subspace_demo.m

subspace_demo.m

```
format compact; format shorte; close

n=10;

A =rand(n,n); A=A'*A; [V,D]=eig(A); for k=1:n; D(k,k)=1+n-k; end;
D(5,5)=1.8;
D(5,5)=6.8;
D(6,6)=2.3/2; D(7,7)=2/2; D(8,8)=1.3/2; D(9,9)=1.1/2; D(10,10)=1/2;
A=V*D*inv(V);

Z=rand(n,4); [Q,R]=qr(Z);

for k=1:100
  Z=A*Q;
  [Q,R]=qr(Z,0);
  ek(k)=abs(R(4,4)-d(4)); kk(k)=k;
end;

[d(3) d(4) d(5)]

r4=abs(d(4)/d(3));
r5=abs(d(5)/d(4));
semilogy(kk,ek,'r.-',kk,r4.^(2*kk),'g-',kk,r5.^(2*kk),'b-')

title('Simultaneous Iteration, 10x10 matrix, 4 vectors','fontsize',14)
xlabel('Iteration, k','fontsize',14); ylabel('Error','fontsize',14);
aleg=legend('|\lambda_4-R_{44}|',...
            '|\lambda_4/\lambda_3|^{2k}',...
            '|\lambda_5/\lambda_4|^{2k}');
leg = findobj(aleg,'type','text'); set(leg,'FontSize',18)
```

subspace_demo.m



Simultaneous Iteration

for $k = 1, 2, \ldots$

$$Z = AQ$$

$$QR = Z \qquad \text{(Normalize } \mathbf{z}_1\text{, orthonormalize } \mathbf{z}_j,\ j > 1.\text{)}$$

end

- **Results:** $Q = [\mathbf{q}_1\, \mathbf{q}_2\, \ldots\, \mathbf{q}_p] \longrightarrow [\mathbf{x}_1\, \mathbf{x}_2\, \ldots\, \mathbf{x}_p]$

  $R_{kk} \longrightarrow \lambda_k$ if first p eigenvalues have distinct modulus.

- This is a Rayleigh quotient scheme.
- Rate of convergence for λ_1 is the same as power iteration with Rayleigh quotient:

$$R = Q^T Z = Q^T A Q$$

$$r_{11} = \frac{\mathbf{q}_1^T A \mathbf{q}_1}{\mathbf{q}_1^T \mathbf{q}_1} = \text{Rayleigh quotient.}$$

Simultaneous Iteration

- $\mathbf{q}_1$ is unaffected by the presence of $\mathbf{q}_j$, $j > 1$.
- Convergence of the entire subspace will depend on the ratio λ_{p+1}/λ_p.
- Convergence of λ_p will also depend on the ratio λ_p/λ_{p-1}.
- Expect convergence of λ_p to scale like s^{2k}, where

$$s \;:=\; \max\left\{\left|\frac{\lambda_{p+1}}{\lambda_p}\right|\left|\frac{\lambda_p}{\lambda_{p-1}}\right|\right\} \;<\; 1,$$

  assuming distinct moduli.

Simultaneous Iteration

- Can extend Z to span all of $\mathcal{R}^n$ (i.e., $p = n$, Z is square).
- In this case, convergence of λ_n will scale like $(\lambda_{n-1}/\lambda_n)^{2k}$.
- This is similar to Rayleigh quotient iteration without shift.
- Can incorporate shift also.
- But there is a more efficient "simultaneous iteration" scheme: QR-iteration.

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

QR Iteration

Recall similarity transformation:

$$A: \quad B \;=\; T^{-1}AT \qquad \textit{same eigenvalues}$$

With $B\underline{z} = \lambda\underline{z}$, we have:

$$\begin{aligned} B\underline{z} = T^{-1}AT\underline{z} &= \lambda\underline{z} \\ AT\underline{z} &= \lambda T\underline{z} \\ A\underline{x} &= \lambda\underline{x}, \qquad \text{with } \underline{x} := T\underline{z} \end{aligned}$$

Starting with $A_0 = A$, we consider a sequence of similarity transformations:

$$A_k \;=\; Q_k^T\, A_{k-1}\, Q_k \;=\; Q_k^{-1}\, A_{k-1}\, Q_k$$

QR Iteration

Start with $A_0 = A$.

- Alg. QR:

for $k = 1, 2, \ldots$

$$\begin{array}{rcl} Q_k R_k & = & A_{k-1} \\ A_k & = & R_k Q_k \end{array} \quad \left| \quad \begin{array}{rcl} R_k & = & Q_k^T A_{k-1} \\ A_k & = & Q_k^T A_{k-1} Q_k \end{array} \right. \quad \left| \quad \begin{array}{rcl} Q_k & = & A_{k-1} R_k^{-1} \\ A_k & = & R_k A_{k-1} R_k^{-1} \end{array} \right.$$

end

- Net result is similarity transformation

$$A_k = (Q_k^T \, Q_{k-1}^T \cdots Q_1^T) \, A \, (Q_1 \, Q_2 \cdots Q_k)$$

- Eigenvalues and symmetry are preserved.
- Can use a different orthogonal matrix $A_0 := Q_0^T A Q_0$ to start.

Comparison of QR and Subspace Iteration

Thus, we have a way of generating $\hat{Q}_k := Q_1 Q_2 \cdots Q_k$ through the following QR iteration:

for $k = 1, 2, \ldots$

$$Q_k R_k = A_{k-1}$$

$$A_k = R_k Q_k$$

$$\hat{Q}_k = \hat{Q}_{k-1} Q_k$$

end

If A is normal, columns approach eigenvectors

QR Iteration

Note that $Q_1 R_1 = A_0 = A$.

$$
\begin{aligned}
A^2 &= Q_1\, R_1\, Q_1\, R_1 \\
&= Q_1\, Q_2\, R_2\, R_1 \;=: \hat{Q}_2 \hat{R}_2
\end{aligned}
$$

$$
\begin{aligned}
A^3 &= Q_1\, R_1\, Q_1\, R_1\, Q_1\, R_1 \\
&= Q_1\, Q_2\, R_2\, Q_2\, R_2\, R_1 \;=: \hat{Q}_2 \hat{R}_2 \\
&= Q_1\, Q_2\, Q_3\, R_3\, R_2\, R_1 \;=: \hat{Q}_3 \hat{R}_3
\end{aligned}
$$

$$
A^k = \hat{Q}_k \hat{R}_k
$$

- Algorithm produces successive powers of A.

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

eig_qr_w_shifts.m < interactive example >

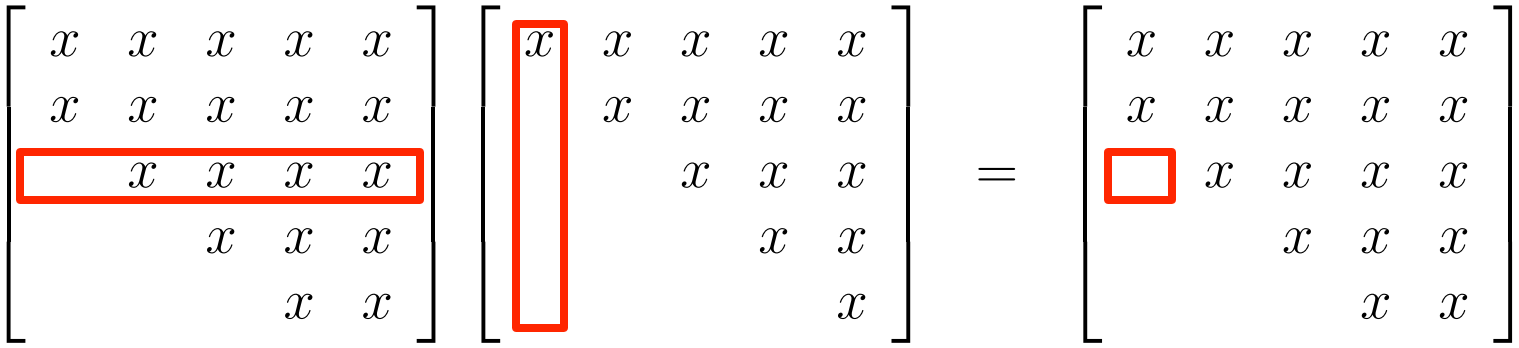
Preliminary Reduction

- 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

Upper Hessenberg X Upper Triangular is Upper Hessenberg

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \\ & & & & x \end{bmatrix} = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \end{bmatrix}$$

Upper Hessenberg X Upper Triangular is Upper Hessenberg



Upper Hessenberg X Upper Triangular is Upper Hessenberg

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \\ & & & & x \end{bmatrix} = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \end{bmatrix}$$

- Same for upper triangular $\times$ upper Hessenberg.
- Start QR iteration with $A_0 := Q_0^T A Q_0$ such that A_0 is upper Hessenberg.
- Then each A_k is upper Hessenberg, and $Q_k R_k$ can be done with Givens rotations in $O(n^2)$ operations, instead of $O(n^3)$.
- With shifted QR, need only $O(n)$ iterations, so total cost is $O(n^3)$.

QR iteration:

for $k = 1, 2, \ldots$

$$Q_k R_k = A_{k-1}$$

$$A_k = R_k Q_k$$

end

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

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

$$\text{or}$$

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

- 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

- Assume in the following that A is a symmetric positive definite matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.
- Suppose we take k iterations of the power method to approximate λ_1:

Algorithm:

$$\mathbf{y}_k = A^k \mathbf{x}$$

$$\lambda = \frac{\mathbf{y}_k^T A \mathbf{y}_k}{\mathbf{y}_k^T \mathbf{y}_k} \approx \lambda_1$$

Code:

$\mathbf{y} = \mathbf{x}$
for $j = 1 : k$
  $\mathbf{y} = A\mathbf{y}$
end
$\lambda = \dfrac{\mathbf{y}^T A \mathbf{y}}{\mathbf{y}^T \mathbf{y}}$

- This approach uses no information from preceding iterates, $\mathbf{y}_{k-1}$, $\mathbf{y}_{k-2}$, $\ldots$ $\mathbf{y}_1$.

Krylov Subspace Methods

- Suppose instead, we seek (for $\mathbf{y} \neq 0$),

$$\begin{aligned} \lambda &= \max_{\mathbf{y} \in \mathcal{K}_{k+1}} \frac{\mathbf{y}^T A \mathbf{y}}{\mathbf{y}^T \mathbf{y}} \\ &= \max_{\mathbf{y} \in \mathbb{P}_k(A)\mathbf{x}} \frac{\mathbf{y}^T A \mathbf{y}}{\mathbf{y}^T \mathbf{y}} \\ &\leq \max_{\mathbf{y} \in \mathbb{R}^n} \frac{\mathbf{y}^T A \mathbf{y}}{\mathbf{y}^T \mathbf{y}} = \lambda_1. \end{aligned}$$

- Here, the **Krylov subspace**

$$\mathcal{K}_{k+1} = \mathcal{K}_{k+1}(A, \mathbf{x}) = \text{span}\{\mathbf{x}\ A\mathbf{x}\ A^2\mathbf{x}\ \ldots\ A^k\mathbf{x}\}$$

  is the space of all polynomials of degree $\leq k$ in A times $\mathbf{x}$:

$$\mathcal{K}_{k+1}(A, \mathbf{x}) = \mathbb{P}_k(A)\mathbf{x}.$$

Krylov Subspace Methods

- Consider the $n \times (k+1)$ matrix

$$V_{k+1} \;=\; \left[\mathbf{x} \;\; A\mathbf{x} \;\; A^2\mathbf{x} \;\; \ldots \;\; A^k\mathbf{x}\right]$$

- Assuming V_{k+1} has full rank (k+1 linearly independent columns), then any $\mathbf{y} \in \mathcal{K}_{k+1}$ has the form

$$\mathbf{y} \;=\; V_{k+1}\,\mathbf{z}, \;\; \mathbf{z} \in \mathbb{R}^{k+1}.$$

- Our optimal Rayleigh quotient amounts to

$$\lambda \;=\; \max_{\mathbf{y}\in\mathcal{K}_{k+1}} \frac{\mathbf{y}^T A\mathbf{y}}{\mathbf{y}^T\mathbf{y}} \;=\; \max_{\mathbf{z}\in\mathbb{R}^{k+1}} \frac{\mathbf{z}^T\left(V_{k+1}^T A\, V_{k+1}\right)\mathbf{z}}{\mathbf{z}^T\left(V_{k+1}^T V_{k+1}\right)\mathbf{z}}$$

Krylov Subspace Methods

- If we had columns $\mathbf{v}_j$ that were orthonormal ($\mathbf{v}_i^T \mathbf{v}_j = \delta_{ij}$), then we'd have $V_{k+1}^T V_{k+1} = I$ and

$$\begin{aligned}
\lambda &= \max_{\mathbf{y} \in \mathcal{R}(V_{k+1})} \frac{\mathbf{y}^T A \mathbf{y}}{\mathbf{y}^T \mathbf{y}} \\
&= \max_{\mathbf{z} \in \mathbb{R}^{k+1}} \frac{\mathbf{z}^T \left(V_{k+1}^T A \, V_{k+1} \right) \mathbf{z}}{\mathbf{z}^T \left(V_{k+1}^T V_{k+1} \right) \mathbf{z}} \\
&= \max_{\mathbf{z} \in \mathbb{R}^{k+1}} \frac{\mathbf{z}^T \, T_{k+1} \, \mathbf{z}}{\mathbf{z}^T \mathbf{z}} \\
&= \mu_1(T_{k+1}) \;\leq\; \lambda_1(A).
\end{aligned}$$

- Here, μ_1 is the maximum eigenvalue of $T_{k+1} := V_{k+1}^T A \, V_{k+1}$.

Krylov Subspace Methods

- This is the idea behind Arnoldi / Lanczos iteration.
- We use information from the entire Krylov subspace to generate optimal eigenpair approximations.
- They require only matrix-vector products (unlike QR iteration, which requires all of A).
- The approximation to λ_1 is given by the eigenvalue μ_1 of the much smaller $(k+1) \times (k+1)$ matrix, T_{k+1}.
- It is the closest approximation to λ_1 out of all possible polynomials of degree k in A and therefore superior (or equal to) the power method.
- Similarly, μ_k is the closest approximation to λ_n.
- The methods produce the best possible approximations (in this subspace) to the extreme eigenvalue/vector pairs.
- Middle eigenpairs are more challenging—must use *shift & invert*.

Krylov Subspace Methods

- Note, for $||\mathbf{z}|| = 1$,

$$\mu_1 \;=\; \max_{||\mathbf{z}||=1} \; \mathbf{z}^T \, T_{k+1} \, \mathbf{z}$$

  corresponds to $\mathbf{z} = \mathbf{z}_1$, so

$$\mu_1 \;=\; \mathbf{z}_1^T \, T_{k+1} \, \mathbf{z}_1 \;=\; \mathbf{z}_1^T \, V_{k+1}^T A \, V_{k+1} \, \mathbf{z}_1 \;\approx\; \lambda_1.$$

- So, corresponding eigenvector approximation for $A\mathbf{x}_1 = \lambda_1 \mathbf{x}_1$ is

$$\mathbf{x}_1 \;\approx\; V_{k+1} \mathbf{z}_1.$$

Krylov Subspace Methods

- **Remark:** Shifting does not improve Lanczos / Arnoldi. WHY?

Krylov Subspace Methods

- **Remark:** Shifting does not improve Lanczos / Arnoldi. WHY?
  - If $p(x) \in \mathbb{P}_k(x)$, so is $p(x+1)$, $p(ax+b)$, *etc.*
  - So: $\mathcal{K}(A, \mathbf{x}) \equiv \mathcal{K}(A + \alpha I, \mathbf{x})$.
  - The spaces are the same and the Krylov subspace projections will find the same optimal solutions.
- Shifting *may help* with conditioning, however, in certain circumstances.
- The essential steps of the algorithms is to construct, step by step, an orthogonal basis for $\mathcal{K}_k$.
- We turn to this for the symmetric (Lanczos) case.

Krylov Subspace Methods

- We start with the symmetric case, known as Lanczos iteration.
- The essence of the method is to construct, step by step, an orthogonal basis for $\mathcal{K}_k$.

$$
\begin{aligned}
&\mathbf{q}_1 \;=\; \mathbf{x}/||\mathbf{x}||;\\
&\text{for } k = 1, \ldots\\
&\quad \mathbf{u} \;=\; A\mathbf{q}_k, \;\; u_0 \;=\; ||\mathbf{u}||\\
&\quad \mathbf{u} \;=\; \mathbf{u} \;-\; P_k\,\mathbf{u}\\
&\quad \beta_k \;=\; ||\mathbf{u}||\\
&\quad \text{if } \beta_k/u_0 < \epsilon, \text{ break}\\
&\quad \mathbf{q}_{k+1} \;=\; \mathbf{u}/\beta_k\\
&\text{end}
\end{aligned}
$$

- Here, $P_k := Q_k Q_k^T$, is the orthogonal projector onto $\mathcal{R}(Q_k)$, so

$$\mathbf{u} \;=\; \mathbf{u} - P_k\mathbf{u} \;=\; \mathbf{u} - \sum_{j=1}^{k} \mathbf{q}_j\,\mathbf{q}_j^T\,\mathbf{u},$$

  which is implemented as modified Gram-Schmidt orthogonalization.

- Q: Why is u_0 useful?

Krylov Subspace Generation

- Notice how the orthogonal subspace is constructed.

$$\begin{aligned} \mathbf{q}_1 &\in \{\mathbf{x}\} \\ \mathbf{q}_2 &\in \{\mathbf{x},\ A\mathbf{x}\} \\ \mathbf{q}_k &\in \{\mathbf{x},\ A\mathbf{x},\ \ldots,\ A^{k-1}\mathbf{x}\} \ =\ \mathcal{K}_k \end{aligned}$$

- In the algorithm, we have

$$\begin{aligned} \mathbf{u} &= A\mathbf{q}_k \\ \mathbf{q}_j^T\mathbf{u} &= \mathbf{q}_j^T A\mathbf{q}_k \\ &= \mathbf{q}_k^T A^T\mathbf{q}_j \ =\ \mathbf{q}_k^T A\mathbf{q}_j \\ &= \mathbf{q}_k^T\mathbf{w}_{j+1},\ \ \mathbf{w}_{j+1} := A\mathbf{q}_j \in \mathcal{K}_{j+1} \end{aligned}$$

- However,

$$\mathbf{q}_k \ \perp\ \mathcal{K}_{j+1}\ \forall\ j+1 \ <\ k.$$

- Therefore

$$\begin{aligned} \mathbf{u} &= \mathbf{u} - P_k\mathbf{u} \\ &= \mathbf{u} \ -\ \mathbf{q}_k\left(\mathbf{q}_k^T A\mathbf{q}_k\right) \ -\ \mathbf{q}_{k-1}\left(\mathbf{q}_{k-1}^T A\mathbf{q}_k\right) \\ &= \mathbf{u} \ -\ \mathbf{q}_k\alpha_k \ -\ \mathbf{q}_{k-1}\beta_{k-1}, \\ \alpha_k &:= \mathbf{q}_k^T A\mathbf{q}_k \ \ \beta_k \ :=\ ||u_{k-1}||. \end{aligned}$$

Lanczos Iteration (A Symmetric)

- The Lancos iteration is:

$$
\begin{aligned}
&\mathbf{q}_1 \;=\; \mathbf{x}/||\mathbf{x}||; \\
&\text{for } k = 1, \ldots \\
&\quad \mathbf{u} \;=\; A\mathbf{q}_k, \;\; u_0 \;=\; ||\mathbf{u}|| \\
&\quad \alpha_k \;=\; \mathbf{q}_k^T \mathbf{u} \\
&\quad \mathbf{u} \;=\; \mathbf{u} \;-\; \alpha_k \mathbf{q}_k \;-\; \beta_{k-1}\mathbf{q}_{k-1} \\
&\quad \beta_k \;=\; ||\mathbf{u}|| \\
&\quad \text{if } \beta_k/u_0 < \epsilon, \text{ break} \\
&\quad \mathbf{q}_{k+1} \;=\; \mathbf{u}/\beta_k \\
&\text{end}
\end{aligned}
$$

Lanczos Iteration (A Symmetric)

- In matrix form,

$$AQ_k \;=\; Q_k T_k \;+\; \beta_{k+1}\mathbf{q}_{k+1}\mathbf{e}_k^T$$

- Or,

$$A\,[\mathbf{q}_1 \;\; \mathbf{q}_2 \;\; \cdots \;\; \mathbf{q}_k] \;=\; [\mathbf{q}_1 \;\; \mathbf{q}_2 \;\; \cdots \;\; \mathbf{q}_k] \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_1 & \beta_1 & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{bmatrix} \;+\; \beta_{k+1}\mathbf{q}_{k+1}\mathbf{e}_k^T .$$

Arnoldi Iteration (A Nonsymmetric)

- Arnoldi iteration is essentially the same as Lanczos, save that we do not get the short term recurrence.

$$
\begin{aligned}
&\mathbf{q}_1 \;=\; \mathbf{x}/||\mathbf{x}||; \\
&\text{for } k = 1, \ldots \\
&\quad \mathbf{u} \;=\; A\mathbf{q}_k, \;\; u_0 \;=\; ||\mathbf{u}|| \\
&\quad \mathbf{u} \;=\; \mathbf{u} \;-\; P_k\,\mathbf{u} \\
&\quad \beta_k \;=\; ||\mathbf{u}|| \\
&\quad \text{if } \beta_k/u_0 < \epsilon, \text{ break} \\
&\quad \mathbf{q}_{k+1} \;=\; \mathbf{u}/\beta_k \\
&\text{end}
\end{aligned}
$$

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 $\boldsymbol{AQ}_n = \boldsymbol{Q}_n\boldsymbol{H}$ yields recurrence

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

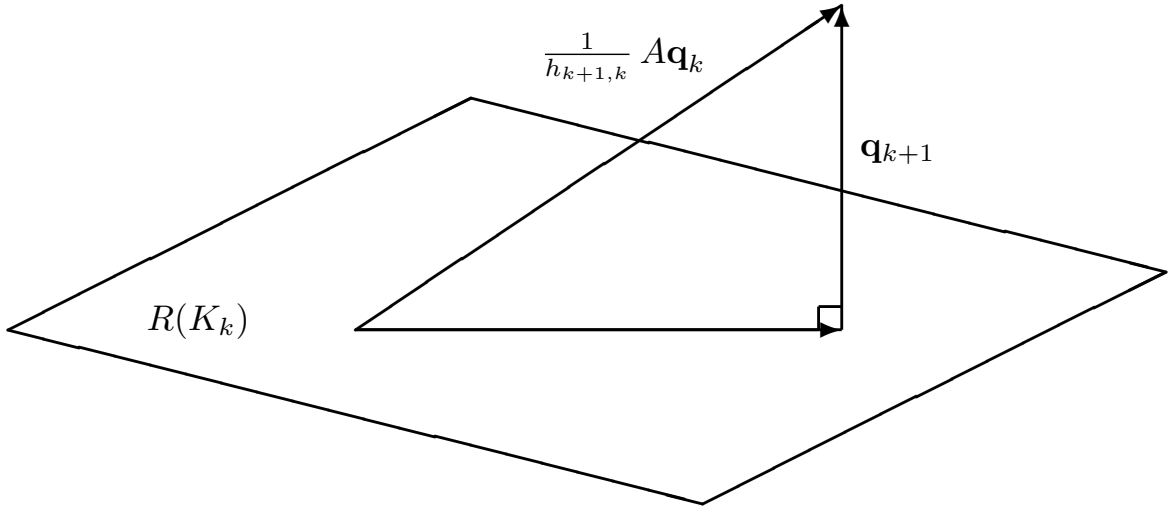
  relating $\boldsymbol{q}_{k+1}$ to preceding vectors $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k$

- 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

Krylov Subspace Projections



- Notice that $A\mathbf{q}_j$ will be in $\mathcal{R}(K_k)$ for all $j < k$.
- $\mathbf{q}_{k+1} \perp \mathcal{R}(K_k)$
- $\mathbf{q}_{k+1} \perp \mathcal{R}(AK_{k-1}) \subset \mathcal{R}(K_k)$

Krylov Subspace and Similarity Transformation

- Consider the rank k matrix

$$K_k \;:=\; \left(\mathbf{x}_0\, A\mathbf{x}_0\, A^2\mathbf{x}_0\, \cdots\, A^{k-1}\mathbf{x}_0\right),$$

and associated Krylov subspace $\mathcal{K}_k := R(K_k)$.

- Krylov subspace methods work with the orthogonal vectors $\mathbf{q}_k \in \mathcal{K}_k$, k=1, 2,. . . , satisfying $QR = K_k$.

- The similarity transformation

$$Q^{-1}AQ \;=\; Q^TAQ = H$$

with entries $h_{ij} = \mathbf{q}_i^T A\mathbf{q}_j$ is upper Hessenberg.

- Proof:
  - $\mathbf{q}_i^T\mathbf{u} = 0$ for any $\mathbf{u} \in K_k$ if $i > k$.
  - $\mathbf{u} = A\mathbf{q}_j \;\in\; K_{j+1}$.
  - $\mathbf{q}_i^T\mathbf{u} = \mathbf{q}_i^T A\mathbf{q}_j = 0$ for $i > j + 1$.

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

$\boldsymbol{x}_0 =$ 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

Modified Gram-Schmidt

Arnoldi Iteration

$x_0 =$ arbitrary nonzero starting vector
$q_1 = x_0 / \|x_0\|_2$
for $k = 1, 2, \ldots$
  $u_k = A q_k$
  for $j = 1$ **to** k
    $h_{jk} = q_j^H u_k$
    $u_k = u_k - h_{jk} q_j$
  end
  $h_{k+1,k} = \|u_k\|_2$
  if $h_{k+1,k} = 0$ **then** stop
  $q_{k+1} = u_k / h_{k+1,k}$
end

Modified Gram-Schmidt

$\underline{u}_k = (I - QQ^T)\,\underline{u}_k$
$= \underline{u}_k - QQ^T\,\underline{u}_k$
$= \underline{u}_k - [h_{1k}\,\underline{q}_1 \;\; h_{2k}\,\underline{q}_2 \ldots h_{kk}\,\underline{q}_k]$

Q: There are two projectors in the lines above. Where are they ??

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 q_k must be orthogonalized against all previous columns of 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 A after relatively few iterations

A reasonable restart choice: current approximate eigenvector

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 $(\boldsymbol{A} - \sigma \boldsymbol{I})\boldsymbol{x} = \boldsymbol{y}$

Optimality of Lanczos, Case A is SPD

- Recall, if A is SPD, then $\mathbf{x}^T A\mathbf{x} > 0 \ \forall \ \mathbf{x} \neq 0$.
- If Q is full rank, then $T := Q^T AQ$ is SPD.

$$(Q\mathbf{y})^T A\,(Q\mathbf{y}) \ = \ \mathbf{y}^T Q^T AQ\mathbf{y} > 0 \ \ \forall \ \mathbf{y} \neq 0.$$

- If A is SPD then $||A||_2 = \lambda_1$ (max eigenvalue). Thus,

$$\lambda_1 \ = \ \max_{\mathbf{x}\neq 0} \frac{\mathbf{x}^T A\mathbf{x}}{\mathbf{x}^T\mathbf{x}} \ = \ \max_{||\mathbf{x}||=1} \mathbf{x}^T A\mathbf{x}.$$

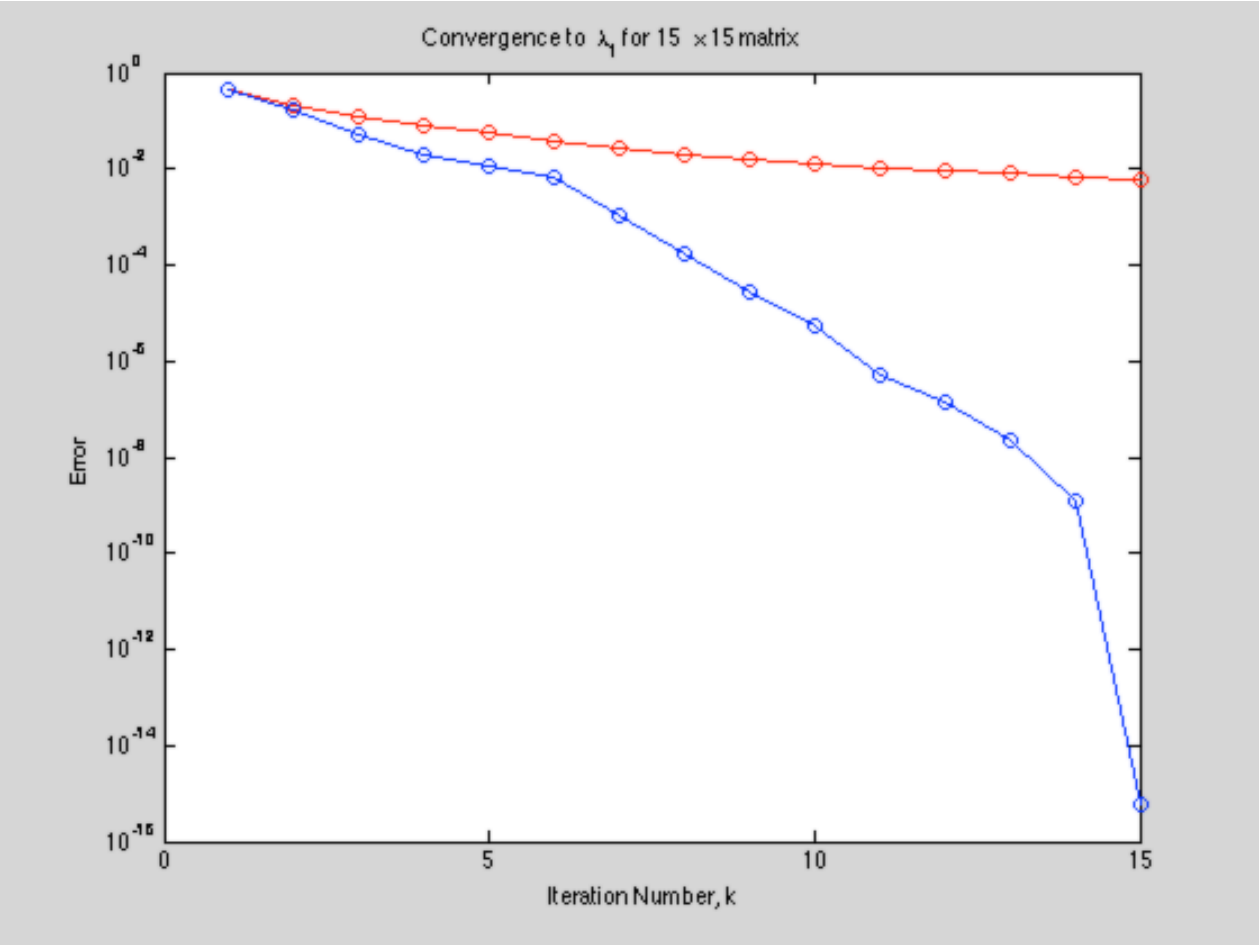
- Let $T\mathbf{y} = \mu\mathbf{y}$. ($T$ is $k \times k$ tridiagonal, $k \ll n$.)

$$\mu_1 \ = \ \max_{||\mathbf{y}||=1} \mathbf{y}^T T\mathbf{y} \ = \ \max_{||\mathbf{y}||=1} \mathbf{y}^T Q^T AQ\mathbf{y} \ \geq \ \mathbf{x}^T A\mathbf{x} \ \ \forall\, \mathbf{x} \in \mathcal{K}_k$$

- Therefore, μ_1 is the closest Rayleigh quotient estimate for all $\mathbf{x} \in \mathcal{K}_k$, $||\mathbf{x}|| = 1$.
- Lanczos is as good as (or much better than) the power method for the same number of matrix-vector products in A.

Matlab Demo

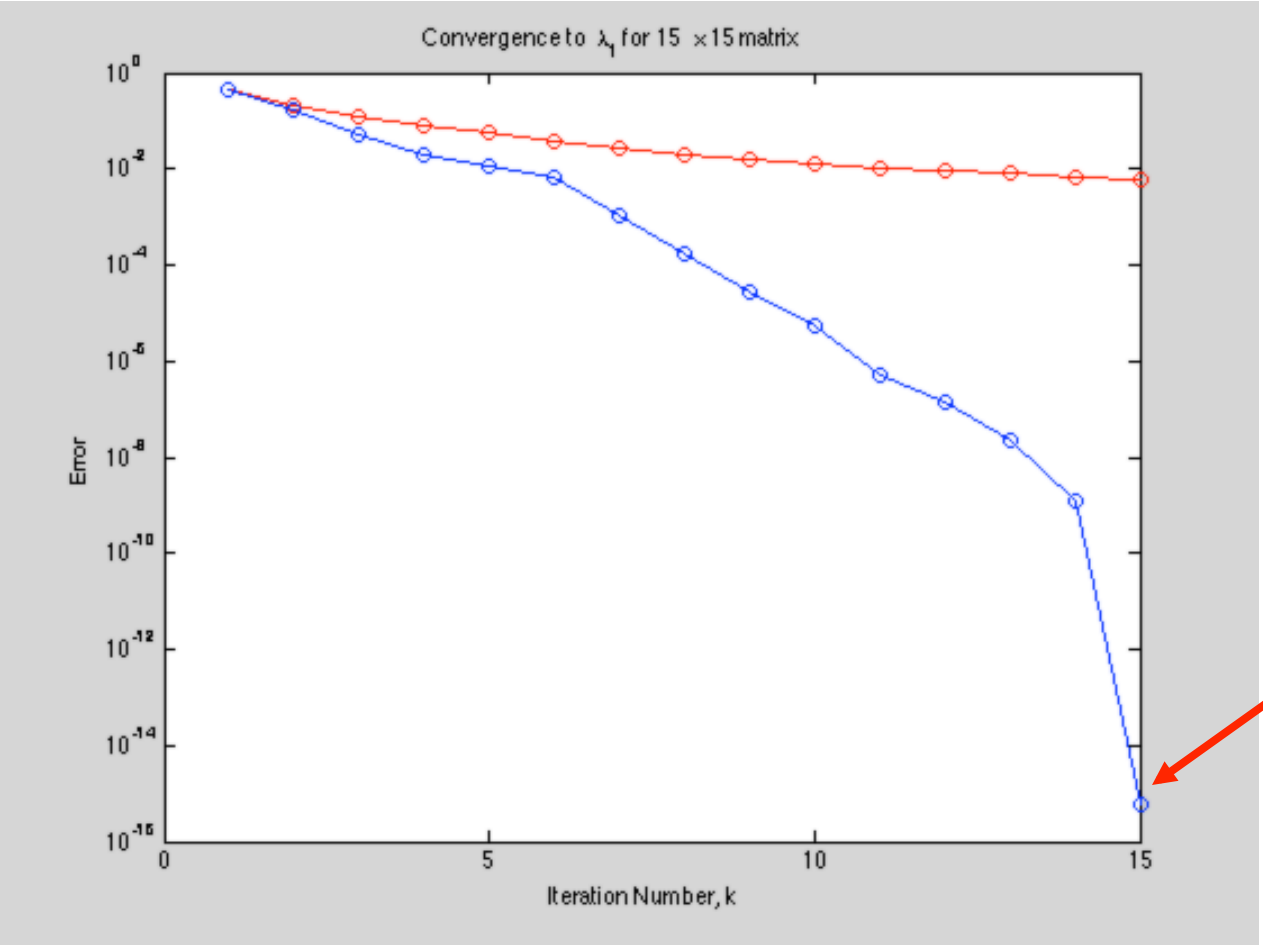
- Lanczos vs Power Iteration



- Lanczos does a reasonable job of converging to extreme eigenvalues.

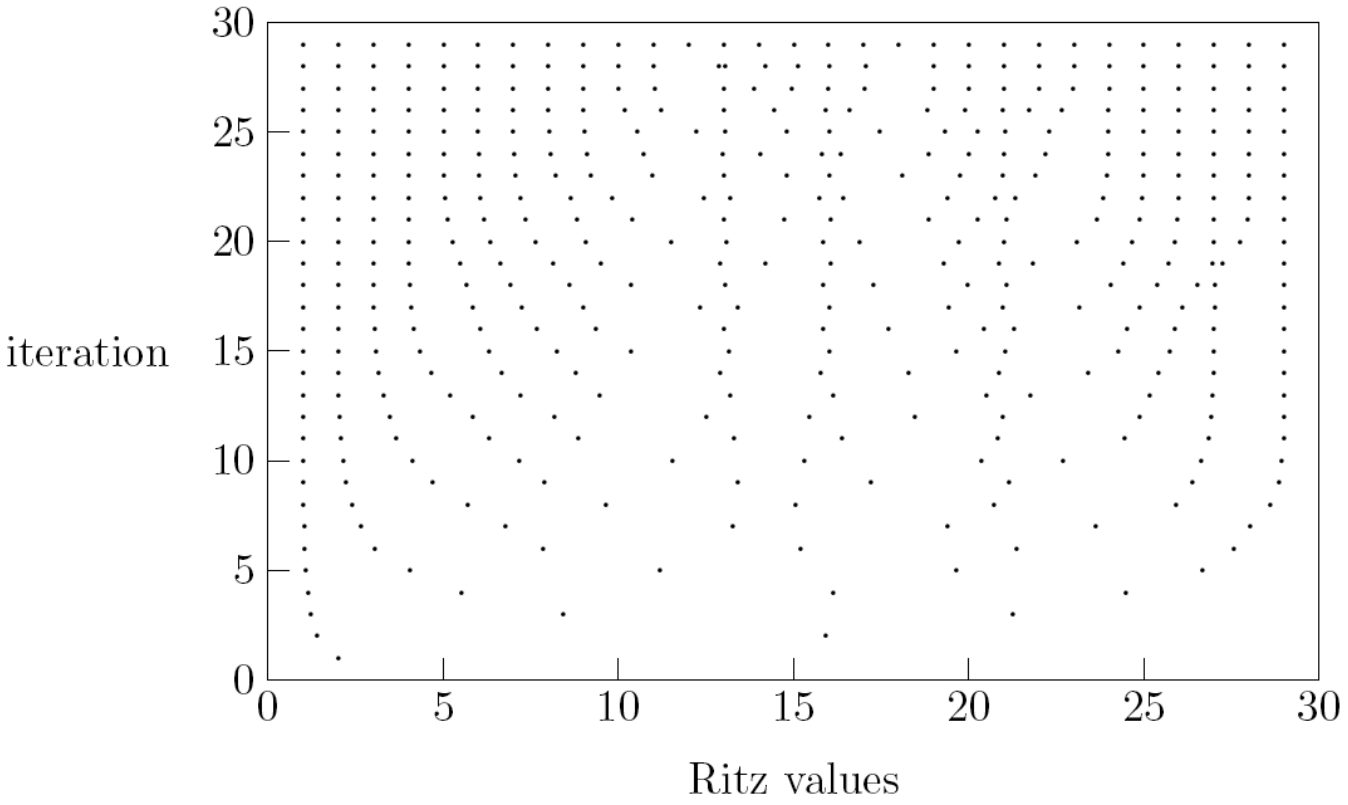
Matlab Demo

- Lanczos vs Power Iteration



Q: What is happening here?

- Lanczos does a reasonable job of converging to extreme eigenvalues.

Example: Lanczos Iteration

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