# Chapter 4: Eigenvalue Problems

# Eigenvalues and Eigenvectors

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

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

- $\lambda$ is the *eigenvalue* and $\mathbf{x}$ is the *eigenvector*

- $\lambda$ (and $\mathbf{x}$) may be complex even if $\mathbf{A}$ is real

- *Spectrum* of $\mathbf{A}$ = set of all eigenvalues $\lambda(\mathbf{A})$

- *Spectral radius* $\rho(\mathbf{A}) = \max\{|\lambda| : \lambda \in \lambda(\mathbf{A})\}$

# Geometric Interpretation

- The matrix-vector product $\hat{\mathbf{v}} = A\mathbf{v}$ stretches or shrinks any vector $\mathbf{v}$ lying in direction of eigenvector $\mathbf{x}$

- Scalar expansion or contraction factor is given by corresponding $\lambda$

- Eigenvalues and eigenvectors lead to simple interpretation of general linear transformations (e.g., as represented by matrix-vector products)

- They are particularly useful when considering iterative processes that can be cast as a sequence of matrix-vector products, such as

$$\mathbf{x}_1 = \mathbf{A}\mathbf{x}_0, \quad \mathbf{x}_2 = \mathbf{A}\mathbf{x}_1, \ \ldots \ , \mathbf{x}_k = \mathbf{A}^k\mathbf{x}_0,$$

- Such sequences are in fact at the core of most of the algorithms used to find the eigenpairs $(\lambda, \mathbf{x})$ of $\mathbf{A}$

# Examples: Eigenvalues and Eigenvectors

- $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ : $\quad \lambda_1 = 1,\ \mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \lambda_2 = 2,\ \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

- $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$ : $\quad \lambda_1 = 1,\ \mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \lambda_2 = 2,\ \mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

- $\mathbf{A} = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}$ : $\quad \lambda_1 = 2,\ \mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \lambda_2 = 4,\ \mathbf{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

# Examples: Eigenvalues and Eigenvectors

- $\mathbf{A} = \begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix}$ : $\quad \lambda_1 = 2,\ \mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \lambda_2 = 1,\ \mathbf{x}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

- $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ : $\quad \lambda_1 = i,\ \mathbf{x}_1 = \begin{bmatrix} 1 \\ i \end{bmatrix}, \quad \lambda_2 = -i,\ \mathbf{x}_2 = \begin{bmatrix} i \\ 1 \end{bmatrix}$

# *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 $1 \times 1$ 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 condition 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 $\lambda$ 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: $\lambda$ and $\mathbf{x}$.
- We refer to $(\lambda, \mathbf{x})$ as an eigenpair, with *eigenvalue* $\lambda$ and *eigenvector* $\mathbf{x}$.

# Solving the Eigenvalue Problem

- The eigenpair satisfies

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

  which is to say,

  - $\mathbf{x}$ is in the null-space of $A - \lambda I$
  - $\lambda$ is chosen so that $A - \lambda I$ has a null-space.

- We thus seek $\lambda$ 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 $A$.

  ***Because $Ax_1 = \lambda x_1$ , i.e., it is simply a stretching of $x_1$***

- 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 $2 \times 2$ 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 $\mathbf{x}_2$.)

# Example 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:

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

- 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 $\lambda$ that has largest modulus.
- This is the main idea behind the ***power method***, which is a common way to find the eigenvector associated with $\max|\lambda|$.

# Characteristic Polynomial

- Equation $\mathbf{Ax} = \lambda\mathbf{x}$ is equivalent to

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

  which has nonzero solution $\mathbf{x}$ *iff* matrix $(\mathbf{A} - \lambda\mathbf{I})$ is singular

- Eigenvalues of $\mathbf{A}$ are roots $\lambda_i$ of *characteristic polynomial*

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

  of degree $n$ in $\lambda$

- *Fundamental Theorem of Algebra* implies that $n \times n$ matrix $\mathbf{A}$ always has $n$ eigenvalues, but they need not be real nor distinct

- Complex eigenvalues of real matrix occur in complex conjugate pairs: If $\lambda = \alpha + i\beta$ is an eigenvalue of a real matrix then so is $\alpha - i\beta$, where $i = \sqrt{-1}$

# Example: Characteristic Polynomial

- Evaluate $\det(\mathbf{A} - \lambda\mathbf{I})$ of earlier example

$$
\begin{aligned}
|\mathbf{A} - \lambda\mathbf{I}| &= \det\left(\begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \\
&= \det\left(\begin{bmatrix} 3-\lambda & -1 \\ -1 & 3-\lambda \end{bmatrix}\right) \\
&= (3-\lambda)(3-\lambda) - (-1)(-1) = \lambda^2 - 6\lambda + 8 = 0
\end{aligned}
$$

- Eigenvalues are

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

# 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*

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

- Roots of polynomial degree $> 4$ cannot always be computed in finite number of steps

- So in general, computation of eigenvalues of matrices of order $> 4$ requires a (theoretically infinite) iterative process

# Example: Companion Matrix, $n = 3$

- Consider companion matrix

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & -c_0 \\ 1 & 0 & -c_1 \\ 0 & 1 & -c_2 \end{bmatrix}$$

- Evaluate determinant of $\mathbf{C} - \lambda\mathbf{I}$

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

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

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

- Roots of resultant monic polynomial, $p(\lambda) = c_0 + c_1\lambda + c_2\lambda^2 + \lambda^3 = 0$, are the 3 eigenvalues, $\lambda_1$, $\lambda_2$, and $\lambda_3$

# 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 a powerful theoretical tool but usually not useful computationally

- In fact, in many cases we use eigenvalue solvers to find the roots of poynomials

# Example: Characteristic Polynomial

- Consider $\mathbf{A} = \begin{bmatrix} 1 & \epsilon \\ \epsilon & 1 \end{bmatrix}$

  with $\epsilon_M < \epsilon < \sqrt{\epsilon_M}$

- Exact eigenvalues of $\mathbf{A}$ are $1 + \epsilon$ and $1 - \epsilon$

- Computing characteristic polynomial in float point arithmetic leads to

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

  which as 1 as a double root

- Thus, eigenvalues cannot be resolved by this method even though they are distinct to working precision

# Multiplicity and Diagonalizability

- *Multiplicity* is number of times root appears when polynomial is written as product of linear factors (e.g., $(1-\lambda)^3(2-\lambda)^2(5-\lambda)$)

***Algebraic Multiplicity***

- Eigenvalue with multiplicity 1 is *simple*

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

***Geometric Multiplicity***

- Nondefective matrix $\mathbf{A}$ has $n$ linearly independent eigenvectors, so it is *diagonalizable*

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

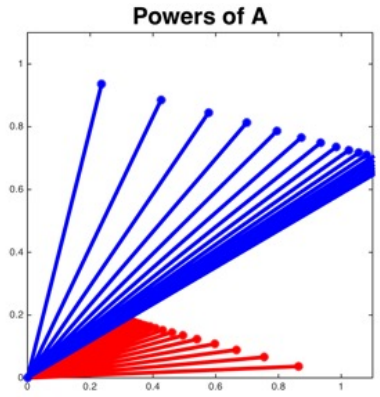
  where $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \cdots \mathbf{x}_n]$ is nonsingular matrix of eigenvectors

- Note: every matrix is $\epsilon$ away from being diagonalizable

# Diagonalization

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

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




pow_a.m

- 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:

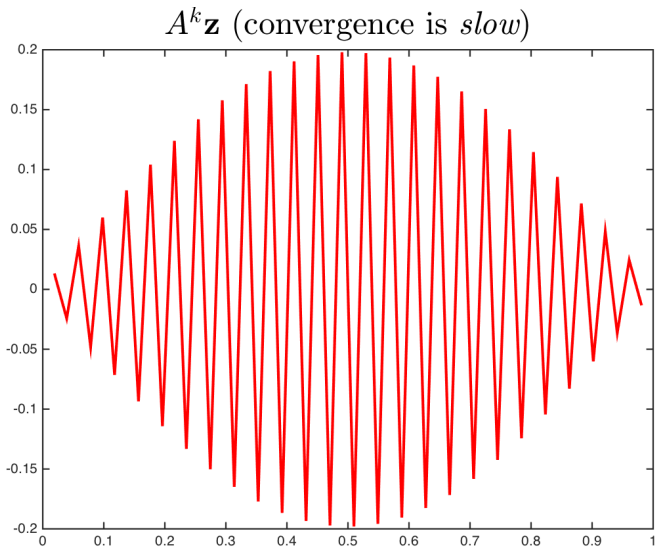
$$(\mathbf{z}_k)_i \;=\; \sin k\pi x_i \;=\; \sin k\pi i\Delta x \qquad \lambda_k \;=\; \frac{2}{\Delta x^2}\left(1-\cos k\pi\Delta x\right)$$

- 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.




```
hdr

n = 50;
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=sin(pi*x);
for k=1:3000;
    z=A*z; z=z/norm(z);
    plot(x,z,'r-',lw,2);
    title('$A^k {\bf z}$ (convergence is {\em slow})',intp,ltx,fs,24);
    drawnow;
    k
end;
```

```
hdr

n=100;


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;
z=sin(pi*x);
for k=1:2000;
    z=A*z; z=z/norm(z);
    plot(x,z,'r-',lw,2); pause;
    k
end;
if -min(z) > max(z); z=-z; end; plot(x,z,'r-'); pause;

[Z,D]=eig(full(A)); zn=Z(:,n);


hold on
zn=zn/norm(zn); if -min(zn) > max(zn); zn=-zn; end;
plot(x,zn,'kx')
sn = sin(n*pi*x); sn=sn/norm(sn);
plot(x,sn,'go')
```

# 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.

*Stopped Here*

## Eigenspaces and Invariant Subspaces

- Eigenvectors can be scaled arbitrarily: if $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$, then $\mathbf{A}(\gamma\mathbf{x}) = \lambda(\gamma\mathbf{x})$ for any scalar $\gamma$, so $\gamma\mathbf{x}$ is also eigenvector corresponding to $\lambda$

- Eigenvectors are usually *normalized* by requiring some norm of eigenvector to be 1 (2-norm is most favored...)

- *Eigenspace* $= \mathcal{S}_\lambda = \{\mathbf{x} : \mathbf{A}\mathbf{x} = \lambda\mathbf{x}\}$

- Subspace $\mathcal{S}$ of $\mathbb{R}^n$ (or $\mathbb{C}^n$) is *invariant* if $\mathbf{A}\mathcal{S} \subseteq \mathcal{S}$

- For eigenvectors $\mathbf{x}_1 \cdots \mathbf{x}_p$ span$([\mathbf{x}_1 \cdots \mathbf{x}_p])$ is invariant subspace

- **Q:** When might invariance fail?

  **A:** In floating-point arithmetic, because of round-off error

# Relevant Properties of Matrices

- Properties of matrix $\mathbf{A}$ 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 | $\mathbf{A}^T\mathbf{A} = \mathbf{A}\mathbf{A}^T = \mathbf{I}$ |
| unitary | $\mathbf{A}^H\mathbf{A} = \mathbf{A}\mathbf{A}^H = \mathbf{I}$ $(\mathbf{A} \in \mathbb{C}^{n\times n})$ |
| symmetric | $\mathbf{A} = \mathbf{A}^T$ |
| skew-symmetric | $\mathbf{A} = -\mathbf{A}^T$ |
| Hermitian | $\mathbf{A} = \mathbf{A}^H$ |
| normal | $\mathbf{A} = \mathbf{A}^H$ |
| normal | $\mathbf{A}^H\mathbf{A} = \mathbf{A}\mathbf{A}^H$ |

# Upper Hessenberg (from Chap03…)

- 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: $\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$ $(\mathbf{A} = \mathbf{A}^T)$

- Skew-Symmetric: $\begin{bmatrix} 0 & 2 \\ -2 & 0 \end{bmatrix}^T = -\begin{bmatrix} 0 & -2 \\ 2 & 0 \end{bmatrix}, \quad (\mathbf{A} = -\mathbf{A}^T)$

# Examples, continued

- Nonsymmetric: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

- 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: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$, $\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}$

- Unitary: $\begin{bmatrix} \frac{i\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & -i\frac{\sqrt{2}}{2} \end{bmatrix}$

- Nonorthogonal $\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$

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

- 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 $\underline{x}_i^H \underline{x}_j = \delta_{ij}$

$$X^T = X^{-1}$$

$$A = XDX^H$$

Normal matrices include

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

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

# 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 the matrix real or complex?
- Is the matrix relatively small and dense, or large and sparse?
- Does the matrix have any special properties such as symmetry, or is it a general matrix?

# 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 (k << n) eigenvectors / eigenvalues.
- Often, there are special properties of A that make it difficult to create A. Instead, work strictly with matrix-vector products

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

# Conditioning of Eigenvalue Problems

- Condition of eigenvalue problem is sensitivity of eigenvalues and eigenvectors to changes in matrix

- Condition of eigenvalue problem is *not* same as conditioning of solution to linear system for same matrix
  - Finding $\lambda = 0$ is a common situation in eigenvalue problems, but indicates a singularity when trying to solve $\mathbf{Ax} = \mathbf{b}$ sensitivity of coefficients of characteristic polynomial

- Different eigenvalues and eigenvectors are not necessarily equally sensitive to perturbations in matrix

# Conditioning of Eigenvalues

- If $\mu$ is eigenvalue of $\mathbf{A} + \mathbf{E}$ of nondefective matrix $\mathbf{A}$, then

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

  where $\lambda_k$ is closest eigenvalue of $\mathbf{A}$ to $\mu$ and $\mathbf{X}$ is the nonsingular matrix of eigenvectors of $\mathbf{A}$

- Absolute condition number of eigenvalues is condition number of matrix of eigenvectors with respect so solving linear equations (e.g., $\mathbf{Xc} = \mathbf{b}$)

- Eigenvalues may be sensitive if eigenvectors are nearly linearly dependent (i.e., matrix is nearly defective)

- For *normal* matrix ($\mathbf{A}\mathbf{A}^H = \mathbf{A}^H\mathbf{A}$), eigenvectors are orthogonal, so eigenvalues are well-conditioned

# Conditioning of Eigenvalues

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

$$|\Delta\lambda| \lesssim \frac{\|\mathbf{y}\|_2 \cdot \|\mathbf{x}\|_2}{|\mathbf{y}^H\mathbf{x}|}\|\mathbf{E}\|_2 = \frac{1}{\cos\theta}\|\mathbf{E}\|_2$$

  where $\mathbf{x}$ and $\mathbf{y}$ are corresponding right and left eigenvectors and $\theta$ is the 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 $\mathbf{Ax} = \lambda\mathbf{x}$ and $\sigma$ is any scalar, then $(\mathbf{A} - \sigma\mathbf{I})\mathbf{x} = (\lambda - \sigma)\mathbf{x}$, so eigenvalues of shifted matrix are shifted eigenvalues of $\mathbf{A}$

- *Inversion*: If $\mathbf{A}$ is nonsingular and $\mathbf{Ax} = \lambda\mathbf{x}$ with $\mathbf{x} \neq \mathbf{0}$, then $\lambda \neq 0$ and $\mathbf{A}^{-1}\mathbf{x} = (1/\lambda)\mathbf{x}$, so eigenvalues of inverse are reciprocals of $\lambda(\mathbf{A})$

- *Powers*: If $\mathbf{Ax} = \lambda\mathbf{x}$, then $\mathbf{A}^k\mathbf{x} = \lambda^k\mathbf{x}$, so eigenvalues of power of matrix are $\lambda^k$

- *Polynomial*: If $\mathbf{Ax} = \lambda\mathbf{x}$, and $p(t)$ is a polynomial, then $p(\mathbf{A})\mathbf{x} = p(\lambda)\mathbf{x}$, so eigenvalues of polynomial in $\mathbf{A}$ are $p(\lambda)$.

# Similarity Transformation

- $\mathbf{B}$ is *similar* to $\mathbf{A}$ if there exists a nonsingular matrix $\mathbf{T}$ such that

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

- Then,

$$\mathbf{B}\mathbf{y} = \lambda\mathbf{y} \implies \mathbf{T}^{-1}\mathbf{A}\mathbf{T}\mathbf{y} = \lambda\mathbf{y} \implies \mathbf{A}\mathbf{T}\mathbf{y} = \lambda\mathbf{T}\mathbf{y}$$

  so $\mathbf{A}$ and $\mathbf{B}$ have the same eigenvalues, and if $\mathbf{y}$ is eigenvector of $\mathbf{B}$, then $\mathbf{x} = \mathbf{T}\mathbf{y}$ is eigenvector of $\mathbf{A}$

- 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, eigenvectors are $\mathbf{X} = \mathbf{I}$
- Diagonal form is desirable in simplifying eigenvalue problems for general matrices by similarity transformations
- But not all matrices are diagonalizable
- Closest one can get, in general, is Jordan form, which is nearly diagonal but may have som nonzero entries on first superdiagonal corresponging to one or more multiple eigenvalues

## Simple non-diagonalizable example, 2 x 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: $\underline{x} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

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

# $3 \times 3$ **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

- Every matrix can be transformed into triangular, *Schur*, form by similarity and diagonal entries of triangular matrix are the eigenvalues

- Eigenvectors are less obvious but straightforward to compute

- If

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

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

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

  is corresponding eigenvector

# Eigenvectors / Eigenvalues of Upper Triangular Matrix

- Suppose $\mathbf{A}$ is upper triangular

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

- Then

$$0 = (\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \underbrace{\begin{bmatrix} \mathbf{U}_{11} & \mathbf{u} & \mathbf{U}_{13} \\ \mathbf{0} & 0 & \mathbf{v}^T \\ \mathbf{O} & \mathbf{0} & \mathbf{U}_{33} \end{bmatrix}}_{(\mathbf{A} - \lambda\mathbf{I})} \underbrace{\begin{bmatrix} \mathbf{y} \\ -1 \\ 0 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \mathbf{U}_{11}\mathbf{y} - \mathbf{u} \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{0}}$$

- Because $\mathbf{U}_{11}$ is nonsingular, can solve $\mathbf{U}_{11}\mathbf{y} = \mathbf{u}$ to find eigenvector $\mathbf{x}$.

# Block Triangular Form

- If

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

  with *square* diagonal blocks, then

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

  so eigenvalue problem breaks into $p$ smaller eigenvalue problems

- *Real* Schur form has $1 \times 1$ diagonal blocks corresponding to real eigenvalues and $2 \times 2$ diagonal blocks corresponding to complex conjugate eigenvalue pairs

# 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.

# Forms Attainable by Similarity: $\mathbf{B} = \mathbf{T}^{-1}\mathbf{AT}$

| **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) |

***Always exists***

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

# *Computing Eigenpairs via Various (Sophisticated!) Forms of Power Iteration*

# Power Iteration

- Simplest method for computing one eigenvalue-eigenvector pair is *power iteration*, which repeatedly multiplies matrix times initial starting vector

- Assume $\mathbf{A}$ has unique eigenvalue of maximum modulus, say $\lambda_1$, with corresponding eigenvector $\mathbf{v}_1$

- Starting from nonzero vector $\mathbf{x}_0$, iteration scheme

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

  converges to multiple of eigenvector $\mathbf{v}_1$ corresponding to *dominant* eigenvalue $\lambda_1$

```
format shorte

D=eye(3); D(1,1) = 4; D(2,2) = 3; D(3,3) = 2;
V=[ 3  4  2 ;
    4  3  2 ;
    0  0  1];

A = V*D*inv(V)

% Power iteration

x=[ 1 ; 1 ; 1 ];

disp(['       k           x1           x2           x3         x1/x1_old']);
disp(['-------------------------------------------------------------------']);

for k=1:20;
    xo = x;
    x=A*x;
    lambda = x(1)/xo(1);
    disp([k x' lambda])
end;
```

```
       k           x1           x2           x3         x1/x1_old
-------------------------------------------------------------------
   1.0000e+00   5.7143e-01   4.2857e-01   2.0000e+00   5.7143e-01
   2.0000e+00  -4.0000e+00  -5.0000e+00   4.0000e+00  -7.0000e+00
   3.0000e+00  -2.6857e+01  -3.2143e+01   8.0000e+00   6.7143e+00
   4.0000e+00  -1.2400e+02  -1.4900e+02   1.6000e+01   4.6170e+00
   5.0000e+00  -5.1371e+02  -6.2529e+02   3.2000e+01   4.1429e+00
   6.0000e+00  -2.0440e+03  -2.5250e+03   6.4000e+01   3.9789e+00
   7.0000e+00  -8.0154e+03  -1.0044e+04   1.2800e+02   3.9214e+00
   8.0000e+00  -3.1324e+04  -3.9749e+04   2.5600e+02   3.9080e+00
   9.0000e+00  -1.2257e+05  -1.5721e+05   5.1200e+02   3.9130e+00
   1.0000e+01  -4.8108e+05  -6.2244e+05   1.0240e+03   3.9249e+00
   1.1000e+01  -1.8947e+06  -2.4686e+06   2.0480e+03   3.9384e+00
   1.2000e+01  -7.4857e+06  -9.8065e+06   4.0960e+03   3.9509e+00
   1.3000e+01  -2.9656e+07  -3.9015e+07   8.1920e+03   3.9616e+00
   1.4000e+01  -1.1774e+08  -1.5541e+08   1.6384e+04   3.9704e+00
   1.5000e+01  -4.6831e+08  -6.1965e+08   3.2768e+04   3.9773e+00
   1.6000e+01  -1.8652e+09  -2.4726e+09   6.5536e+04   3.9828e+00
   1.7000e+01  -7.4363e+09  -9.8722e+09   1.3107e+05   3.9870e+00
   1.8000e+01  -2.9672e+10  -3.9434e+10   2.6214e+05   3.9901e+00
   1.9000e+01  -1.1847e+11  -1.5757e+11   5.2429e+05   3.9926e+00
   2.0000e+01  -4.7321e+11  -6.2978e+11   1.0486e+06   3.9944e+00
```

# Convergence of Power Iteration

- To see why power iteration converges to dominant eigenvector, express starting vector $\mathbf{x}_0$ as linear combination Starting from nonzero vector $\mathbf{x}_0$, iteration scheme

$$\mathbf{x}_0 = \sum_{j=1}^{n} c_j \mathbf{v}_j$$

  where $\mathbf{v}_j$ are eigenvectors of $\mathbf{A}$

- Then

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

$$= \sum_{j=1}^{n} \lambda_j^k c_j \mathbf{v}_j = \lambda_1^k \left[ c_1\mathbf{v}_1 + \sum_{j=2}^{n} \left( \frac{\lambda_j^k}{\lambda_1} \right)^k c_j \mathbf{v}_j \right]$$

- Because $|\lambda_j/\lambda_1| < 1$ for $j > 1$, successively higher powers go to zero, leaving only $\mathbf{v}_1$ component

# 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 $\boldsymbol{x}_k$ from one iteration to next converges to dominant eigenvalue $\lambda_1$
- 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 (i.e., $c_1 = 0$). Not a problem in practice because rounding error usually introduces such a 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 a complex eigenvector

# 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 eath iteration, say, by requiring its largest component to be 1 in modulus, giving iteration scheme

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

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

```
format shorte
D=eye(3); D(1,1) = 4; D(2,2) = 3; D(3,3) = 2;
V=[ 3  4  2 ;
    4  3  2 ;
    0  0  1];
A = V*D*inv(V)

disp(['       k           x1           x2           x3         x1/x1_old']);
disp(['-------------------------------------------------------------------']);

x=[ 1 ; 1 ; 1 ];
for k=1:20;          % Normalized Power iteration
    y=A*x;
    abs_lambda = norm(y,Inf);
    x=y/abs_lambda;
    disp([k x' abs_lambda])
end;
```

```
       k           x1           x2           x3         x1/x1_old
-------------------------------------------------------------------
  1.0000e+00   2.8571e-01   2.1429e-01   1.0000e+00   2.0000e+00
  2.0000e+00  -8.0000e-01  -1.0000e+00   8.0000e-01   2.5000e+00
  3.0000e+00  -8.3556e-01  -1.0000e+00   2.4889e-01   6.4286e+00
  4.0000e+00  -8.3221e-01  -1.0000e+00   1.0738e-01   4.6356e+00
  5.0000e+00  -8.2157e-01  -1.0000e+00   5.1177e-02   4.1965e+00
  6.0000e+00  -8.0950e-01  -1.0000e+00   2.5347e-02   4.0382e+00
  7.0000e+00  -7.9807e-01  -1.0000e+00   1.2744e-02   3.9777e+00
  8.0000e+00  -7.8804e-01  -1.0000e+00   6.4404e-03   3.9577e+00
  9.0000e+00  -7.7967e-01  -1.0000e+00   3.2568e-03   3.9550e+00
  1.0000e+01  -7.7289e-01  -1.0000e+00   1.6451e-03   3.9594e+00
  1.1000e+01  -7.6753e-01  -1.0000e+00   8.2963e-04   3.9659e+00
  1.2000e+01  -7.6334e-01  -1.0000e+00   4.1768e-04   3.9726e+00
  1.3000e+01  -7.6011e-01  -1.0000e+00   2.0997e-04   3.9784e+00
  1.4000e+01  -7.5764e-01  -1.0000e+00   1.0543e-04   3.9833e+00
  1.5000e+01  -7.5576e-01  -1.0000e+00   5.2881e-05   3.9872e+00
  1.6000e+01  -7.5434e-01  -1.0000e+00   2.6505e-05   3.9903e+00
  1.7000e+01  -7.5326e-01  -1.0000e+00   1.3277e-05   3.9926e+00
  1.8000e+01  -7.5245e-01  -1.0000e+00   6.6477e-06   3.9944e+00
  1.9000e+01  -7.5184e-01  -1.0000e+00   3.3273e-06   3.9958e+00
  2.0000e+01  -7.5138e-01  -1.0000e+00   1.6650e-06   3.9969e+00
```

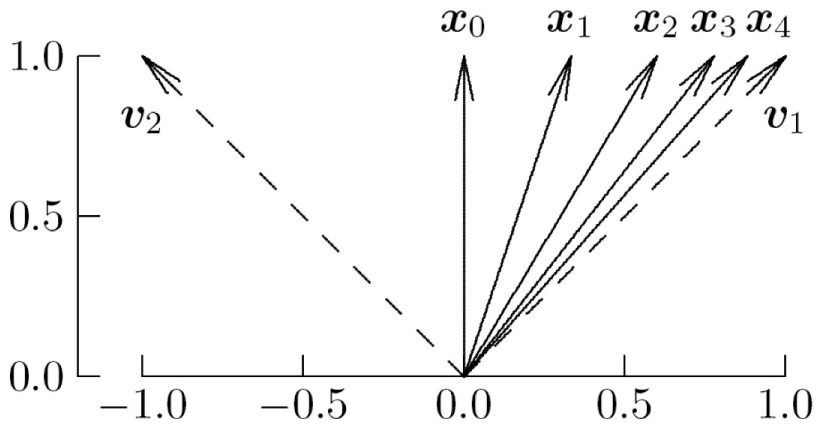
# Example: Normalized Power Iteration

- 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




- Initial vector $\mathbf{x}_0 = \mathbf{v}_1 + \mathbf{v}_2$ contains equal components in $\mathbf{v}_1$ and $\mathbf{v}_2$ (dashed arrows)

- Repeated multiplication by $\mathbf{A}$ causes component in $\mathbf{v}_1$ (corresponding to larger eigenvalue 2) to dominate, so sequence of vectors $\mathbf{x}_k$ converges to $\mathbf{v}_1$

# Convergence Rate of Power Iteration

Convergence rate of power iteration depends on relative separation of $\lambda_1$ and $\lambda_2$.

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

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

***In preceding text examples, ratio is ½, so error is reduced by ½ with each iteration.***

***demo10/err_norm.m***

```
format shorte
D=eye(3); D(1,1) = 4; D(2,2) = 3; D(3,3) = 2;
V=[ 3  4  2 ;
    4  3  2 ;
    0  0  1];
A = V*D*inv(V)

lam_max = 4;

disp(['       k           x1           x2           x3          x1/x1_old']);
disp(['-------------------------------------------------------------------']);

x=[ 1 ; 1 ; 1 ]; est=0;
for k=1:40;          % Normalized Power iteration
    y=A*x;
    abs_lambda = norm(y,Inf);
    x=y/abs_lambda;
    err = abs(4-abs_lambda);
    disp([k x' abs_lambda err est])
    est = 0.75*err;
end;
```

```
      k          x1          x2          x3         x1/x1_old
-------------------------------------------------------------------
  1.0000e+00   2.8571e-01   2.1429e-01   1.0000e+00   2.0000e+00   2.0000e+00            0
  2.0000e+00  -8.0000e-01  -1.0000e+00   8.0000e-01   2.5000e+00   1.5000e+00   1.5000e+00
  3.0000e+00  -8.3556e-01  -1.0000e+00   2.4889e-01   6.4286e+00   2.4286e+00   1.1250e+00
  4.0000e+00  -8.3221e-01  -1.0000e+00   1.0738e-01   4.6356e+00   6.3556e-01   1.8214e+00
  5.0000e+00  -8.2157e-01  -1.0000e+00   5.1177e-02   4.1965e+00   1.9655e-01   4.7667e-01
  6.0000e+00  -8.0950e-01  -1.0000e+00   2.5347e-02   4.0382e+00   3.8154e-02   1.4741e-01
  7.0000e+00  -7.9807e-01  -1.0000e+00   1.2744e-02   3.9777e+00   2.2348e-02   2.8615e-02
  8.0000e+00  -7.8804e-01  -1.0000e+00   6.4404e-03   3.9577e+00   4.2344e-02   1.6761e-02
  9.0000e+00  -7.7967e-01  -1.0000e+00   3.2568e-03   3.9550e+00   4.4979e-02   3.1758e-02
  1.0000e+01  -7.7289e-01  -1.0000e+00   1.6451e-03   3.9594e+00   4.0631e-02   3.3734e-02
  1.1000e+01  -7.6753e-01  -1.0000e+00   8.2963e-04   3.9659e+00   3.4076e-02   3.0473e-02
  1.2000e+01  -7.6334e-01  -1.0000e+00   4.1768e-04   3.9726e+00   2.7436e-02   2.5557e-02
  1.3000e+01  -7.6011e-01  -1.0000e+00   2.0997e-04   3.9784e+00   2.1555e-02   2.0577e-02
  1.4000e+01  -7.5764e-01  -1.0000e+00   1.0543e-04   3.9833e+00   1.6673e-02   1.6166e-02
  1.5000e+01  -7.5576e-01  -1.0000e+00   5.2881e-05   3.9872e+00   1.2768e-02   1.2505e-02
  1.6000e+01  -7.5434e-01  -1.0000e+00   2.6505e-05   3.9903e+00   9.7127e-03   9.5762e-03
  1.7000e+01  -7.5326e-01  -1.0000e+00   1.3277e-05   3.9926e+00   7.3552e-03   7.2845e-03
  1.8000e+01  -7.5245e-01  -1.0000e+00   6.6477e-06   3.9944e+00   5.5531e-03   5.5164e-03
  1.9000e+01  -7.5184e-01  -1.0000e+00   3.3273e-06   3.9958e+00   4.1839e-03   4.1649e-03
  2.0000e+01  -7.5138e-01  -1.0000e+00   1.6650e-06   3.9969e+00   3.1479e-03   3.1380e-03
  2.1000e+01  -7.5104e-01  -1.0000e+00   8.3298e-07   3.9976e+00   2.3661e-03   2.3609e-03
  2.2000e+01  -7.5078e-01  -1.0000e+00   4.1668e-07   3.9982e+00   1.7773e-03   1.7746e-03
  2.3000e+01  -7.5058e-01  -1.0000e+00   2.0841e-07   3.9987e+00   1.3344e-03   1.3330e-03
  2.4000e+01  -7.5044e-01  -1.0000e+00   1.0423e-07   3.9990e+00   1.0016e-03   1.0008e-03
  2.5000e+01  -7.5033e-01  -1.0000e+00   5.2125e-08   3.9992e+00   7.5156e-04   7.5116e-04
  2.6000e+01  -7.5025e-01  -1.0000e+00   2.6066e-08   3.9994e+00   5.6388e-04   5.6367e-04
  2.7000e+01  -7.5019e-01  -1.0000e+00   1.3034e-08   3.9996e+00   4.2302e-04   4.2291e-04
  2.8000e+01  -7.5014e-01  -1.0000e+00   6.5177e-09   3.9997e+00   3.1733e-04   3.1727e-04
  2.9000e+01  -7.5010e-01  -1.0000e+00   3.2590e-09   3.9998e+00   2.3803e-04   2.3799e-04
  3.0000e+01  -7.5008e-01  -1.0000e+00   1.6296e-09   3.9998e+00   1.7854e-04   1.7852e-04
  3.1000e+01  -7.5006e-01  -1.0000e+00   8.1483e-10   3.9999e+00   1.3391e-04   1.3390e-04
  3.2000e+01  -7.5004e-01  -1.0000e+00   4.0742e-10   3.9999e+00   1.0044e-04   1.0043e-04
  3.3000e+01  -7.5003e-01  -1.0000e+00   2.0372e-10   3.9999e+00   7.5332e-05   7.5329e-05
  3.4000e+01  -7.5002e-01  -1.0000e+00   1.0186e-10   3.9999e+00   5.6500e-05   5.6499e-05
  3.5000e+01  -7.5002e-01  -1.0000e+00   5.0930e-11   4.0000e+00   4.2376e-05   4.2375e-05
  3.6000e+01  -7.5001e-01  -1.0000e+00   2.5465e-11   4.0000e+00   3.1783e-05   3.1782e-05
  3.7000e+01  -7.5001e-01  -1.0000e+00   1.2733e-11   4.0000e+00   2.3837e-05   2.3837e-05
  3.8000e+01  -7.5001e-01  -1.0000e+00   6.3664e-12   4.0000e+00   1.7878e-05   1.7878e-05
  3.9000e+01  -7.5001e-01  -1.0000e+00   3.1832e-12   4.0000e+00   1.3409e-05   1.3409e-05
  4.0000e+01  -7.5000e-01  -1.0000e+00   1.5916e-12   4.0000e+00   1.0056e-05   1.0056e-05
```

# Power Iteration with Shift

- Convergence rate of power iteration depends on ratio $|\lambda_2/\lambda_1|$, where $\lambda_2$ is eigenvalue having second largest modulus

- May be possible to choose shift, $\mathbf{A} - \sigma\mathbf{I}$ such that

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

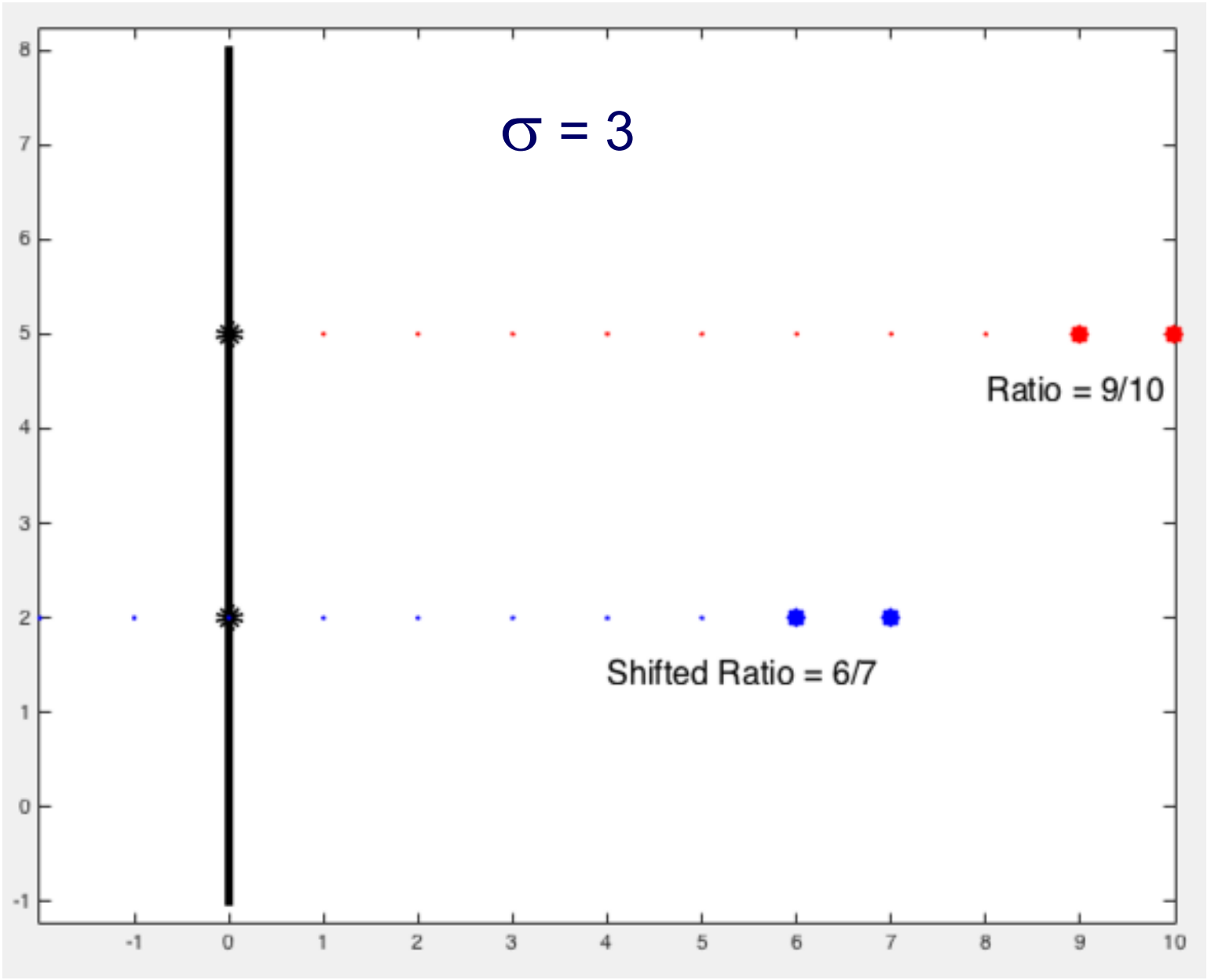
  so convergence is accelerated

- Shift $\sigma$ must then be added to result to obtain eigenvalue of original matrix

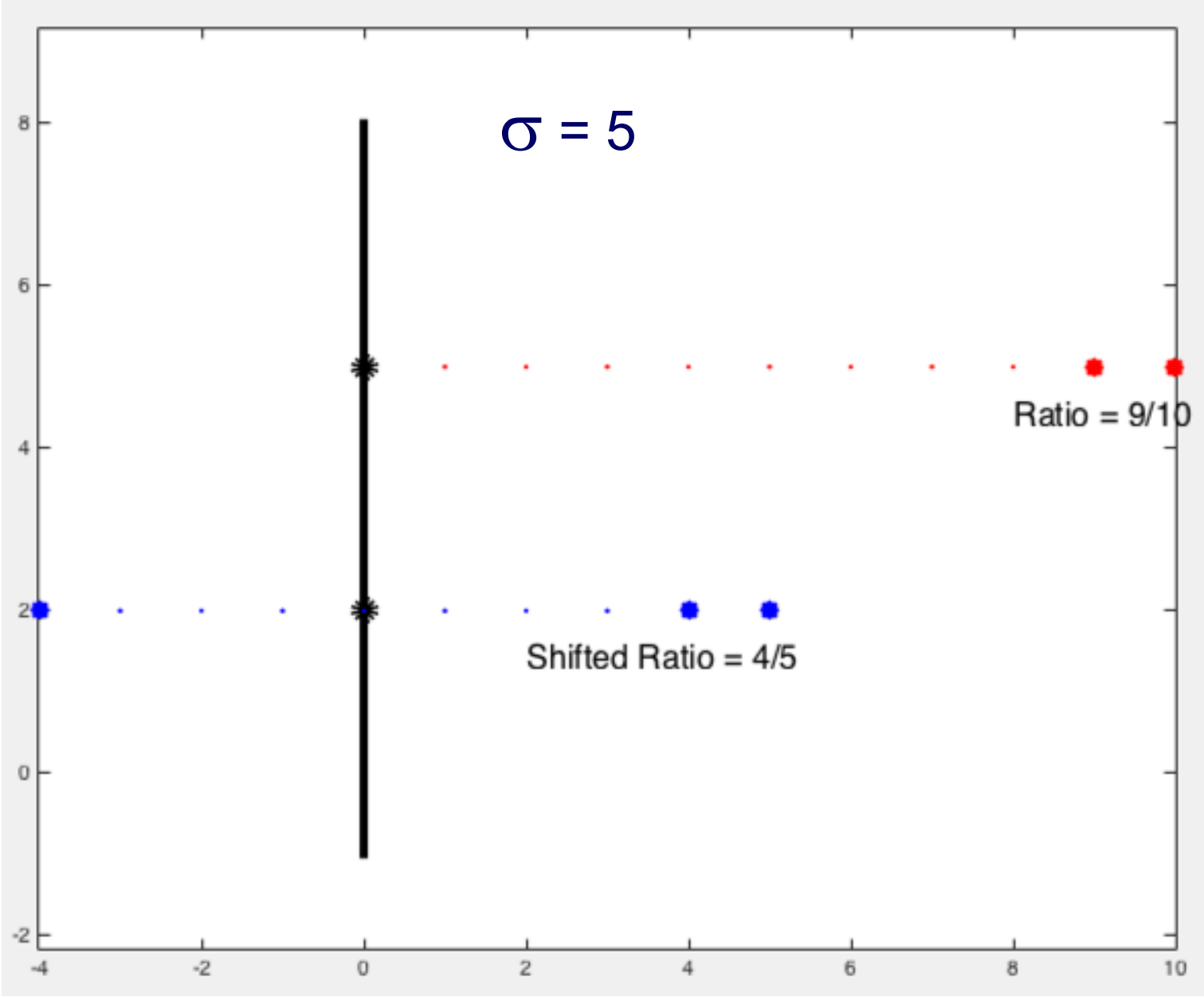
- In earlier example, if we pick $\sigma = 1$ (equal to other eigenvalue), ratio becomes zero and method converges in one iteration

- In general, we would not be able to make such fortuitous choice, but shifts can be extremely useful in some contexts, particularly with *inverse iteration*, which we will see later

# Eigenvalue Ratios:  Shifted and Unshifted



# Eigenvalue Ratios: Shifted and Unshifted



# Eigenvalue Ratios: Shifted and Unshifted

```
%
%  Shifted power demo
%

n=10;
X=rand(n,n); [X,R]=qr(X);
Lam=[n:-1:1]; L=diag(Lam);

A=X*L*X';


z=[1:n]'; z=z/norm(z); zu=z; zs=z;

s=4; I=eye(n);
H=A-s*I; err_u = 1; err_s = 1;
for k=1:400;

    yu = A*zu; lam_u = norm(yu,Inf); zu=yu/lam_u;

    ys = H*zs; lam_s = norm(ys,Inf)+s; zs=ys/norm(ys,Inf);

    err_ul=err_u; err_u = abs(Lam(1)-lam_u)/(Lam(1)); ru = err_ul/err_u;
    err_sl=err_s; err_s = abs(Lam(1)-lam_s)/(Lam(1)); rs = err_sl/err_s;

    disp([k err_u err_s ru rs]), pause

end;
```

# 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 is required rather than largest, can make use of fact that eigenvalues of $\mathbf{A}^{-1}$ are reciprocals of $\lambda(\mathbf{A})$, so smallest eigenvalue of $\mathbf{A}$ is largest eigenvalue of $\mathbf{A}^{-1}$

- This leads to *inverse iteration* scheme,

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

  which is equivalent to power iteration applied to $\mathbf{A}^{-1}$

- Inverse of $\mathbf{A}$ not computed explicitly, but factorization of $\mathbf{A}$ used to solve system of equations at each iteration

- Can of course reuse the LU factors so that cost of successive iterations is relatively low because we only need to solve triangular systems $\mathbf{L}\mathbf{z} = \mathbf{x}_{k-1}$ and $\mathbf{U}\mathbf{y}_k = \mathbf{z}$

# Inverse Iteration, continued

- Inverse iteration converges to eigenvector corresponding to *smallest* eigenvalue of $\mathbf{A}$

- Eigenvalue obtained is dominant eigenvalue of $\mathbf{A}^{-1}$ and hence its reciprocal is smallest eigenvalue of $\mathbf{A}$ 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)

*Again, the error is reduced by ~½ on each iteration.*

# Inverse Iteration with Shift

- As before, shifting strategy, working with $\mathbf{A} - \sigma\mathbf{I}$ for some scalar $\sigma$, can greatly improve performance

- Inverse iteration is particularly useful for computing eigenvector corresponding to approximate eigenvalue because it converges rapidly when applied to shifted matrix $\mathbf{A} - \hat{\lambda}\mathbf{I}$, where $\hat{\lambda}$ is approximate eigenvalue

- Inverse iteration is also useful for computing eigenvalue closest to given value $\beta$ because, if $\beta$ 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 Rate of Convergence

- Assume $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_{n-1}| > |\lambda_n|$

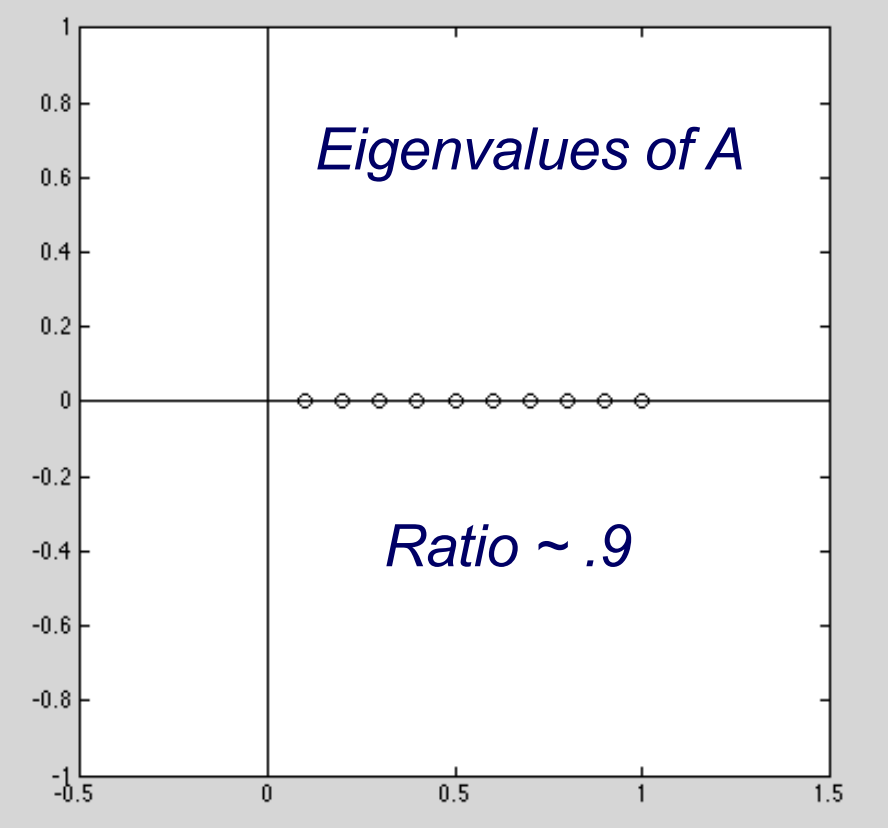
- $$\begin{aligned}
\mathbf{x}^k &= \mathbf{A}^{-k}\mathbf{x}^0 \\
&= \sum_{j=1}^{n} \mathbf{x}_j \left(\frac{1}{\lambda_j}\right)^k c_j \\
&= \left(\frac{1}{\lambda_n}\right)^k \left[\mathbf{x}_n c_n + \mathbf{x}_{n-1}\left(\frac{\lambda_n}{\lambda_{n-1}}\right)^k c_{n-1} + \sum_{j=1}^{n-2} \mathbf{x}_j \left(\frac{\lambda_n}{\lambda_j}\right)^k c_j\right] \\
&\sim \left(\frac{1}{\lambda_n}\right)^k \left[\mathbf{x}_n c_n + \mathbf{x}_{n-1}\left(\frac{\lambda_n}{\lambda_{n-1}}\right)^k c_{n-1}\right]
\end{aligned}$$

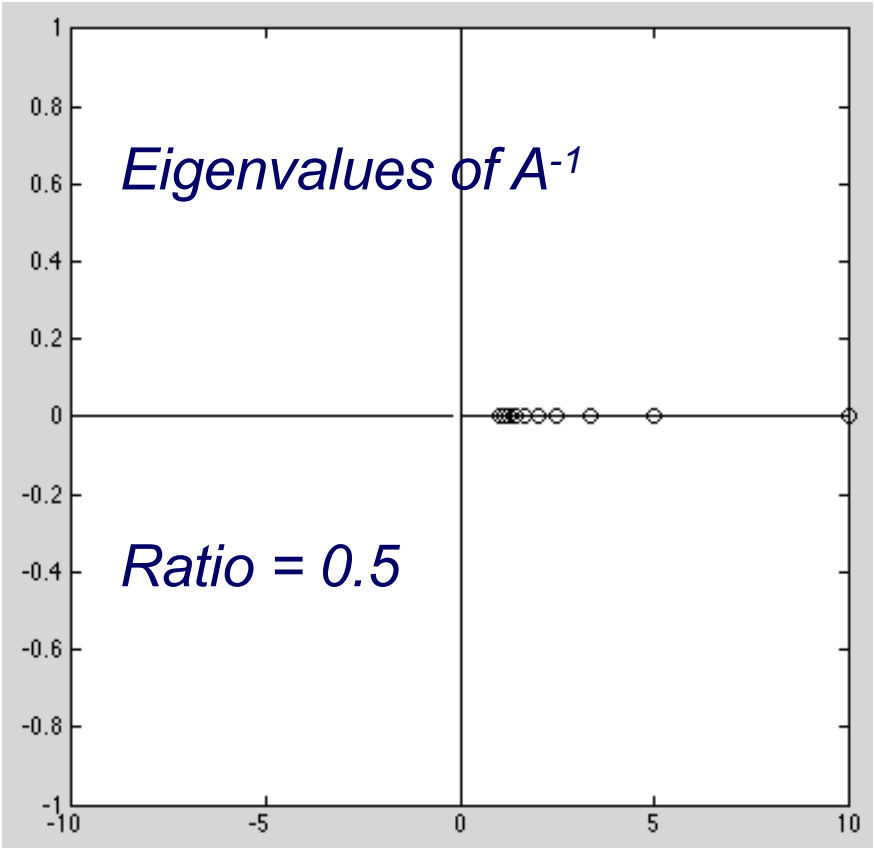
- Rate of convergence is controlled by ratio $|\lambda_n/\lambda_{n-1}|$

*eig_inv_demo.m*

# Inverse Iteration Illustration

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



# Rate of Convergence for Inverse Iteration with Shift

- Assume $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_{n-1}| > |\lambda_n|$

- Let $\mathbf{M} := \mathbf{A} - \sigma I$

  $\mu_j := \lambda_j - \sigma$, and

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

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

- Using current approximation to $\lambda_l$ 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.

# Stopped Here

# Rayleigh Quotient

- Given approximate eigenvector $\mathbf{x}$ for real matrix $\mathbf{A}$, determining best estimate for corresponding $\lambda$ can be considered as a $n \times 1$ LLSQ approximation problem,

$$\mathbf{x}\lambda \approx \mathbf{A}\mathbf{x}$$

- From normal equation, $\mathbf{x}^T\mathbf{x}\lambda = \mathbf{x}^T\mathbf{A}\mathbf{x}$, LLSQ solution is

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

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

# Example: Rayleigh Quotient

- Rayleigh quotient can accelerate convergence of iterative methods such as power iteration because Rayleigh quotient $\mathbf{x}_k^T\mathbf{A}\mathbf{x}_k/\mathbf{x}_k^T\mathbf{x}_k$ gives better approximation to $\lambda$ that basic power iteration

- For previous example from text 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 |

# Convergence of Rayleigh Quotient, Symmetric A

- Suppose $\mathbf{x} = \mathbf{x}_1 + \epsilon\mathbf{w}$ with $\mathbf{x}_1$ the unit 2-norm eigenvector of $\mathbf{A}$ and $\mathbf{w} \perp \mathbf{x}_1$ also having unit 2-norm

- $\|\mathbf{x} - \mathbf{x}_1\| = \epsilon$

- Consider Rayleigh quotient,

$$\frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{\mathbf{x}^T\mathbf{x}} = \frac{\mathbf{x}_1^T\mathbf{A}\mathbf{x}_1}{\mathbf{x}_1^T\mathbf{x}_1 + \epsilon^2} + \frac{\epsilon^2\mathbf{w}^T\mathbf{A}\mathbf{w}}{1 + \epsilon^2}$$

$$= \lambda_1 + O(\epsilon^2)$$

## Some Rayleigh Quotient Properties

- View $r(\mathbf{x})$ as a function of $\mathbf{x} \in \mathbb{R}^n$,

$$r(\mathbf{x}) \;=\; \frac{\mathbf{x}^T A\mathbf{x}}{\mathbf{x}^T\mathbf{x}}.$$

- Then *gradient* of $r$ is

$$\frac{\partial}{\partial x_j} r(\mathbf{x}) \;=\; \frac{2}{\mathbf{x}^T\mathbf{x}} \left(A\mathbf{x} - r(\mathbf{x})\mathbf{x}\right)_j$$

$$\nabla r(\mathbf{x}) \;=\; \frac{2}{\mathbf{x}^T\mathbf{x}} \left(A\mathbf{x} - r(\mathbf{x})\mathbf{x}\right).$$

- If $\mathbf{x}$ is an eigenvector of $A$ then $r(\mathbf{x}) = \lambda$.
- Moreover, $\nabla r(\mathbf{x}) = 0$.

- **Main result:** If $\mathbf{q}_j$ is the $j$th eigenvector of $A$, then

$$|r(\mathbf{x}) - r(\mathbf{q}_j)| \;=\; O(\|\mathbf{x} - \mathbf{q}_j\|^2) \;\text{ as } \mathbf{x} \longrightarrow \mathbf{q}_j.$$

# Rayleigh Quotient Iteration

- For approximate eigenvector, Rayleigh quotient yields good estimate for corresponding $\lambda$

- Moreover, inverse iteration converges rapidly to eigenvector if approximate eigenvalue is used as shift

- Combining these two ideas leads to *Rayleigh Quotient Iteration*,

$$\begin{aligned} \sigma_k &= \mathbf{x}_k^T \mathbf{A} \mathbf{x}_k / \mathbf{x}_k^T \mathbf{x}_k \\ (\mathbf{A} - \sigma_k \mathbf{I}) \mathbf{y}_{k+1} &= \mathbf{x}_k \\ \mathbf{x}_{k+1} &= \mathbf{y}_{k+1} / \|\mathbf{y}_{k+1}\|_2 \end{aligned}$$

- If we start from $\mathbf{x}_0$ with $\|\mathbf{x}_0\|_2 = 1$, the normalization $\mathbf{x}_k^T \mathbf{x}_k$ is not needed.

# Rayleigh Quotient Iteration, continued

- Rayleigh quotient iteration is especially effective for symmetric matrices and usually converges very rapidly (e.g., just one or two iterations may suffice)

- 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

- Factorization overhead can be amortized by taking several iterations by reusing $\mathbf{LU}$ factors between refactorizations. Fixed point iteration will still be convergent

- If $\mathbf{A}$ is tridiagonal, factorization and solve cost is only $\approx 8n$ operations

# Example: Rayleigh Quotient Iteration

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

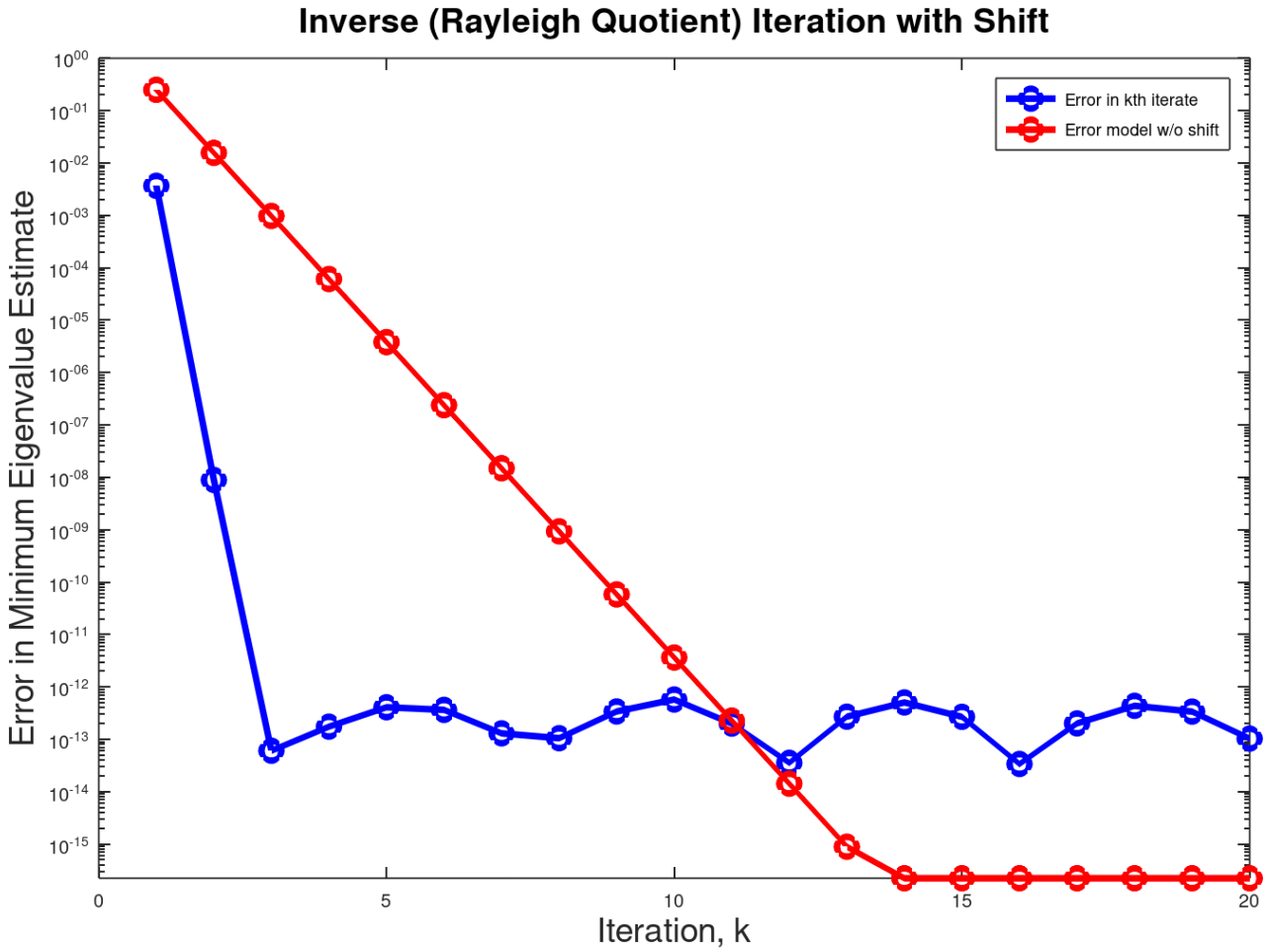
| $k$ | $x_k^T$ | | $\sigma_k$ |
|---|---|---|---|
| 0 | 0.807 | 0.397 | 1.896 |
| 1 | 0.924 | 1.000 | 1.998 |
| 2 | 1.000 | 1.000 | 2.000 |

demo10/eig_shift_invert.m

```
hdr; close all;

n = 50; h = 1/(n+1); e = ones(n,1);
A = spdiags([-e 2*e -e],-1:1, n,n)/(h*h);  % SPD tridiag(-1,2,-1)/h^2
I = speye(n);

z=rand(n,1); sigma = 0;
format compact;
for k=0:20;
    x=z/norm(z);
    M=A-sigma*I;
    z=M\x;       % Inverse iteration, with shift
    mu = 1./(x'*z);
    lambda = mu+sigma;
    sigma  = lambda;
    if k>0; kk(k)=k; lk(k)=lambda; end;
end;

emin = 2*(1-cos(pi*h))/(h*h);
err = abs(emin-lk);

semilogy(kk,err,'bo-',lw,2,kk,max((.25).^(2*kk-1),eps),'ro-',lw,2);
title('Inverse (Rayleigh Quotient) Iteration with Shift',fs,18);
xlabel('Iteration, k',fs,18);
ylabel('Error in Minimum Eigenvalue Estimate',fs,18);
legend('Error in kth iterate','Error model w/o shift','location','northeast')

axis([0 20 eps 1])
```

*eig_inv_demo.m*
*eig_shift_invert.m*

```
hdr; close all;

n = 50; h = 1/(n+1); e = ones(n,1);
A = spdiags([-e 2*e -e],-1:1, n,n)/(h*h);  % SPD tridiag(-1,2,-1)/h^2
I = speye(n);

z=rand(n,1); sigma = 0;
format compact;
for k=0:20;
    x=z/norm(z);
    M=A-sigma*I;
    z=M\x;       % Inverse iteration, with shift
    mu = 1./(x'*z);
    lambda = mu+sigma;
    sigma  = lambda;
    if k>0; kk(k)=k; lk(k)=lambda; e
end;

emin = 2*(1-cos(pi*h))/(h*h);
err = abs(emin-lk);

semilogy(kk,err,'bo-',lw,2,kk,max((.
title('Inverse (Rayleigh Quotient) I
xlabel('Iteration, k',fs,18);
ylabel('Error in Minimum Eigenvalue
legend('Error in kth iterate','Error

axis([0 20 eps 1])
```



# Deflation

- After eigenvalue $\lambda_1$ and corresponding $\mathbf{x}_1$ has been computed, additional eigenvalues can be computed by *deflation*, which effectively removes known eigenvalue

- Let $\mathbf{H}$ be any nonsingular matrix such that $\mathbf{H}\mathbf{x}_1 = \alpha\mathbf{e}_1$, scalar multiple of first column of identity matrix (Householder transformation is a good candidate for $\mathbf{H}$)

- Then similarity transformation determined by $\mathbf{H}$ transforms $\mathbf{A}$ into form

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

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

## Deflation, continued

- Thus, we can work with $\mathbf{B}$ to compute next eigenvalue $\lambda_2$
- Moreover, if $\mathbf{y}_2$ is eigenvector of $\mathbf{B}$ corresponding to $\lambda_2$, then

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

  is eigenvector corresponding to $\lambda_2$ for original matrix $\mathbf{A}$, provided $\lambda_1 \neq \lambda_2$

- Process can be repeated to find additional eigenpairs

# 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 $\lambda_2$.

# Matrix-Free Deflation – Symmetric A

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.

**assuming $\mathbf{x}_1$ has unit 2-norm.*

# Deflation on the Fly – *Subspace Iteration*

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

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

for $k = 1, 2, \ldots$

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

$\mathbf{y}_2$ orthogonal to $\mathbf{y}_1$

- $\mathbf{y}_1$ converges to $\mathbf{x}_1$ (*standard power iteration*).
- $(\mathbf{y}_1\ \mathbf{y}_2)$ converge to $span(\mathbf{x}_1\ \mathbf{x}_2)$.
- $\mathbf{y}_2$ converges to $\mathbf{x}_2$ if $A$ is symmetric or $\mathbf{x}_2 \perp \mathbf{x}_1$.

*subspace.m*

- $\mathbf{z}_1^T\mathbf{x}_1 \longrightarrow \lambda_1$ and $\mathbf{z}_2^T\mathbf{x}_2 \longrightarrow \lambda_2$.

# Simultaneous Iteration

- Simplest method for computing many eigenpairs is *simultaneous iteration*, which repeatedly multiplies matrix $\mathbf{A}$ times matrix of initial starting vectors.

- Starting from $n \times p$ matrix $\mathbf{X}_0$ of rank $p$, iteration scheme is

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

- span($\mathbf{X}_k$) converges to invariant subspace determined by $p$ dominant eigenvalues of $\mathbf{A}$ provide $|\lambda_p| > |\lambda_{p+1}|$

- Also called *subspace iteration*

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

# Simultaneous Iteration

- Suppose $A$ is symmetric ( $\longrightarrow$ eigenvectors are orthogonal)
- Start with $n \times p$ random matix $Q$.

for $k = 1, 2, \ldots$

$Z = AQ$

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

end

*Very Important!*

- **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 $\lambda_1$ 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 $\lambda_{p+1}/\lambda_p$.
- Convergence of $\lambda_p$ will also depend on the ratio $\lambda_p/\lambda_{p-1}$.
- Expect convergence of $\lambda_p$ to scale like $s^{2k}$, where

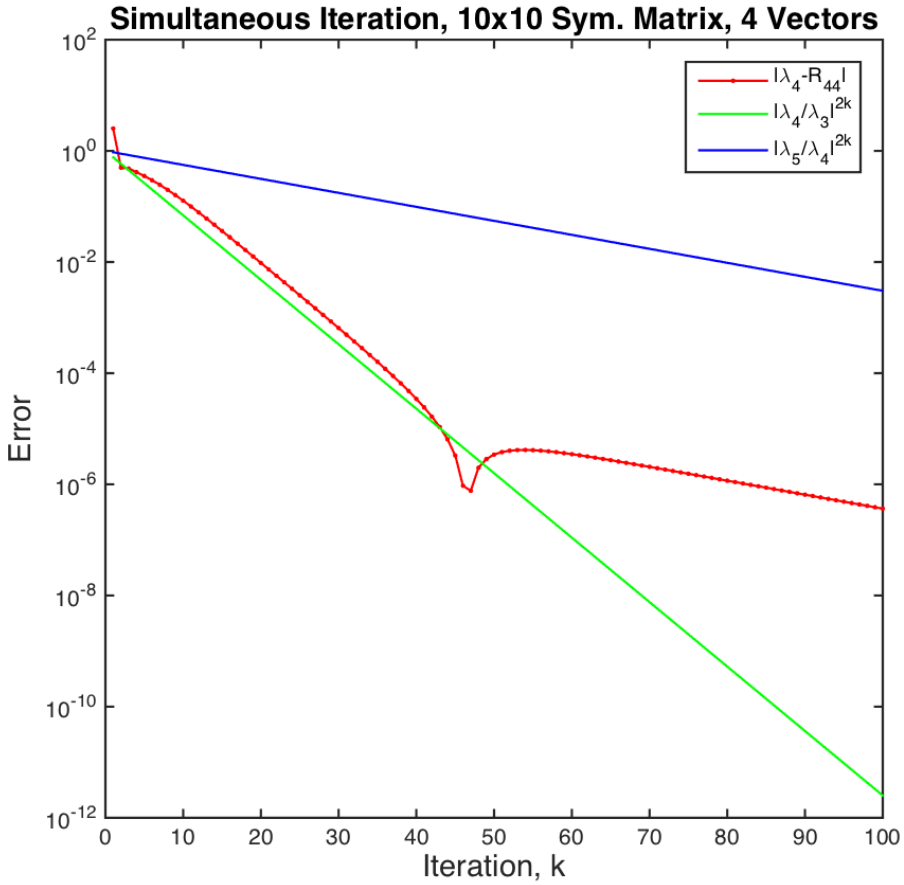
$$s \;:=\; \max\left\{\left|\frac{\lambda_{p+1}}{\lambda_p}\right|\left|\frac{\lambda_p}{\lambda_{p-1}}\right|\right\} \;<\; 1,$$

  assuming distinct moduli and orthogonal eigenvectors.

Matlab Demo:

- *subspace_demo.m*
- *n=10, p=4*

# subspace_demo.m




- In this example,

$$.875 = \frac{\lambda_4}{\lambda_3} < \frac{\lambda_5}{\lambda_4} = .9714,$$

  which implies asymptotic rate will be dominated by the most slowly decaying value, i.e.,

$$|\lambda_4 - R_{44}| \sim C \left(\frac{\lambda_5}{\lambda_4}\right)^{2k} = C\,(0.9714)^{2k}.$$

- With carefully chosen (yet random) $A$ and $Z$, the behavior at the beginning is more like $(0.875)^{2k}$ because, e.g., the initial $\mathbf{x}_5$ component in $Z$ might be small.

# 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)
```

# Simultaneous Iteration

- Can extend $Z$ to span all of $\mathcal{R}^n$ (i.e., $p = n$, $Z$ is square).
- In this case, convergence of $\lambda_n$ will scale like $|\lambda_n/\lambda_{n-1}|^{2k}$.
- This is similar to Rayleigh quotient iteration without shift.
- Can incorporate shift also, so convergence scales like:

$$\left| \frac{\lambda_n - \sigma}{\lambda_{n-1} - \sigma} \right|^{2k},$$

***shift4.m demo***

  which can be made fast with $\sigma = R_{nn} \approx \lambda_n$.

# Important Role of $QR$ in Orthogonal Iteration

- Consider $$Z = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \underbrace{Q_1 R}_{\textit{Reduced } QR} = \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix} \begin{bmatrix} \sqrt{3} \end{bmatrix}$$

$$= \underbrace{QR}_{\textit{Full } QR} = \begin{bmatrix} \underbrace{Q_1}_{\mathcal{R}(Z)} & \underbrace{Q_2}_{\mathcal{R}(Z)^\perp} \end{bmatrix} \begin{bmatrix} R \\ O \end{bmatrix}.$$

- For a *rank-deficient matrix*, $Z$, full $QR$ produces two subspaces,

$$\mathcal{R}(Q_1) = \mathcal{R}(Z) \text{ and } \mathcal{R}(Q_2) = \mathcal{R}(Z)^\perp.$$

- Consider now what happens with orthogonal iteration in the case where an $n \times n$ matrix $A$ has a single 0 eigenvalue, $\lambda_n$.

$$
\begin{aligned}
Z &= \text{rand}(n, n) \\
QR &= Z \qquad \Longrightarrow \quad \mathcal{R}(Q) = \mathcal{R}(Z) \\
Z &= AQ \qquad \Longrightarrow \quad \mathcal{R}(Z) = \mathcal{R}(A) \\
QR &= Z \qquad \Longrightarrow \quad \mathcal{R}(Q_1) = \mathcal{R}(A), \;\; \mathcal{R}(Q_2) = \mathcal{R}(A)^{\perp}.
\end{aligned}
$$

- After the second pass, orthogonal iteration has discovered $\mathbf{x}_n$, the eigenvector associated with $\lambda_n = 0$!

**qrtest.m demo**

# qrtest.m demo

```
hdrd

A=[ 1 0 0 ;
    1 0 0 ;
    1 0 0 ];
a=A(:,1);

[Q,R]=qr(a), [Q,R]=qr(A)

pause

A=[ 1 1 0 ;
    2 1 0 ;
    3 1 0 ];
[Q,R]=qr(A)

disp(' '), disp('Test symmetric matrix with zero eigenvalue'), disp(' ')

V=[ 3 2 1 ;
   -4 1 2 ;
    0 1 3 ];
[q,R]=qr(V); Q=[ q(:,2) q(:,3) q(:,1) ] d=[ 2 1 0 ]; D=diag(d);

disp(' '), disp('A has nontrivial nullspace'), disp(' ')
A=Q*D*Q'


disp(' '), disp('Orthogonal Iteration'), disp(' '), pause

Z=rand(3,3)
for k=0:2; disp(' '),k
    [Q,R]=qr(Z)
    Z = A*Q
    pause
end;
```

# Orthogonal Iteration

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

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

  where $\hat{\mathbf{Q}}_k \mathbf{R}_k$ is *reduced* QR factorization of $\mathbf{X}_{k-1}$

- $\mathcal{R}(\mathbf{X}_k)$ and $\mathcal{R}(\hat{\mathbf{Q}}_k)$ converge to invariant subspace, span$\{\mathbf{x}_1 \ \ldots \ \mathbf{x}_p\}$, and are related by

$$\mathbf{X}_k = \mathbf{A}\hat{\mathbf{Q}}_k = \hat{\mathbf{Q}}_k \mathbf{B}_k \iff \mathbf{B}_k = \hat{\mathbf{Q}}_k^T \mathbf{A}\hat{\mathbf{Q}}_k$$

- $\mathbf{B}_k$ converges to block-upper triangular (Schur) form if eigenvalues are complex, or to upper-triangular matrix if eigenvalues are distinct
- If $\mathbf{A}$ is symmetric then $\hat{\mathbf{Q}}_k \longrightarrow$ eigenvectors of $\mathbf{A}$; otherwise additional (straighforward) work is required to retrieve eigenvectors

# Deflation in Case of Orthogonal Iteration with Shifts

- As we will see, QR iteration with shifts will produce an upper-block triangular similarity transformation of $\mathbf{A}$ of the form

$$\mathbf{B}_k = \begin{bmatrix} b_{1,1} & \dots & b_{1,n-1} & b_{1,n} \\ b_{2,1} & \dots & b_{2,n-1} & b_{2,n} \\ \vdots & & \vdots & \vdots \\ b_{n-1,1} & \dots & b_{n-1,n-1} & b_{n-1,n} \\ 0 & \dots & 0 & b_{n,n} \end{bmatrix}$$

- Incorporated with shifts, $b_{n,n}$ rapidly converges to *simple* $\lambda_n$

- Afterwards, simply work on the leading $(n-1) \times (n-1)$ block, applying shifts of form $\sigma = b_{n-1,n-1}$ (or more sophisticated choice if target eigenvalue is not simple)

- Because system morphs into upper block triangular form, explicit deflation is not needed

# Reducing Work per Eigenvalue: $QR$ iteration

- We now have the basic ingredients of a fast eigensolver.
- However, each matrix-vector product, $AQ$, and each $QR$ factorization requires $O(n^3)$ work.
- If we assume a fixed number of iterations (e.g., $< 4$) for each of the $n$ eigenvalues, then the total work per eigenvalue is $O(n^3)$, giving a total complexity of $O(n^4)$.
- $QR$ iteration is an approach that allows us to reduce the work per eigenvalue to $O(n^2)$.

# Orthogonal Iteration $\longrightarrow$ QR Iteration

- Goals: Incorporate shifts to accelerate convergence
  Reduce cost per iteration to $O(n^2)$ instead of $O(n^3)$

- Orthogonal iteration with computation of $\mathbf{B}_k$ with $p = n$ and $\mathbf{X}_0 = A$

$$\begin{aligned} \hat{\mathbf{Q}}_k \mathbf{R}_k &= \mathbf{X}_{k-1} \\ \mathbf{X}_k &= \mathbf{A}\hat{\mathbf{Q}}_k, \\ \mathbf{B}_k &= \hat{\mathbf{Q}}_k^T \mathbf{A} \hat{\mathbf{Q}}_k = \hat{\mathbf{Q}}_k^T \mathbf{X}_k \longrightarrow \begin{cases} \text{upper triangular via} \\ \text{similarity transform} \end{cases} \end{aligned}$$

- Consider next the *QR iteration*, starting with $\mathbf{X}_0 = \mathbf{A}$

$$\mathbf{Q}_1\mathbf{R}_1 = \mathbf{A}, \qquad \mathbf{B}_1 = \mathbf{Q}_1^T\mathbf{A}\mathbf{Q}_1 = \mathbf{Q}_1^T(\mathbf{Q}_1\mathbf{R}_1)\mathbf{Q}_1 = \mathbf{R}_1\mathbf{Q}_1$$

$$\begin{aligned} \mathbf{Q}_2\mathbf{R}_2 = \mathbf{B}_1, \qquad \mathbf{B}_2 &= \mathbf{Q}_2^T\mathbf{B}_1\mathbf{Q}_2 = \mathbf{Q}_2^T\mathbf{Q}_1^T\mathbf{A}\mathbf{Q}_1\mathbf{Q}_2 = \text{similarity transform} \\ &= \mathbf{Q}_2^T(\mathbf{Q}_2^T\mathbf{R}_2)\mathbf{Q}_2 = \mathbf{R}_2\mathbf{Q}_2 \end{aligned}$$

$$\mathbf{Q}_3\mathbf{R}_3 = \mathbf{B}_2, \qquad \mathbf{B}_3 = \mathbf{R}_3\mathbf{Q}_3 = \underbrace{\mathbf{Q}_3^T\mathbf{Q}_2^T\mathbf{Q}_1^T}_{\hat{\mathbf{Q}}_3^T}\,\mathbf{A}\,\underbrace{\mathbf{Q}_3\mathbf{Q}_2\mathbf{Q}_1}_{\hat{\mathbf{Q}}_3}$$

# QR Iteration

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

$$\mathbf{A}_k = \hat{\mathbf{Q}}_k \mathbf{A} \hat{\mathbf{Q}}_k$$

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

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

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

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

  and form reverse product

$$\mathbf{A}_k = \mathbf{R}_k \mathbf{Q}_k \; (= \mathbf{Q}_k^T \mathbf{A}_{k-1} \mathbf{Q}_k)$$

- Successive $\mathbf{A}_k$ unitarily similar to $\mathbf{A}_{k-1}$

- Diagonal entries (or eigenvalues of diagonal blocks) of $\mathbf{A}_k$ converge to eigenvalues of $\mathbf{A}$

- If $\mathbf{A}$ is symmetric,
  - then symmetry is preserved by QR iteration, so $\mathbf{A}_k$ converges to matrix that is both triangular and symmetric, hence diagonal.
  - $\mathbf{Q}_k$ converges to matrix of corresponding eigenvectors

## Qualitative Interpretation of $QR$ Iteration

- Notice that when we run the $QR$-iteration we should generate

$$\hat{Q}_k \;:=\; \hat{Q}_{k-1}Q_k \;=\; Q_1Q_2\cdots Q_k,$$

  as it is the matrix that contains the approximate eigenvectors.

- Recall that since

$$A_k \;=\; R_kQ_k = Q_k^T(Q_kR_k)Q_k = Q_k^TA_{k-1}Q_k,$$

  we have $A_k = \hat{Q}_k^TA\hat{Q}_k$.

- If $A$ has $n$ orthogonal eigenvectors, then as $\hat{Q}_k \longrightarrow X$, the matrix of eigenvectors, we have

$$A_k \;=\; \hat{Q}_k^TA\hat{Q}_k \;\longrightarrow\; X^{-1}AX \;\longrightarrow\; D,$$

  the matrix of eigenvalues.

# QR Iteration with Shifts

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

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

  where $\sigma_k$ is rough approximation to eigenvalue

- Good shift can be determined by computing eigenvalues of $2 \times 2$ submatrix in lower right of corner matrix (corresponding to $\approx \lambda_{\min}$)

- Convergence is quadratic and possibly cubic, so only 1 or 2 iterations required per eigenvalue

- Then deflate and iterate on remaining $(n-k) \times (n-k)$ block for $k = 1, \ldots, n-1$

- Because $\mathbf{A}_k$ is upper Hessenberg, using Givens rotations for QR factorization leads to only $O(n^2)$ work per iteration

- Number of iterations is $\approx n$

# Preliminary Reduction

- Notice that each QR iteration requires both a QR factorization and an evaluation of the reverse product

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

- For a general matrix $A_{k-1}$ the work is nominally $O(n^3)$

- If $A_{k-1}$ were upper Hessenberg, however, then QR factorization could be effected in $\approx 2n^2 \ll n^3$ operations using Givens rotations

$$\mathbf{R}_k = \mathbf{Q}_k^T(\mathbf{A}_{k-1} - \sigma_k\mathbf{I})$$

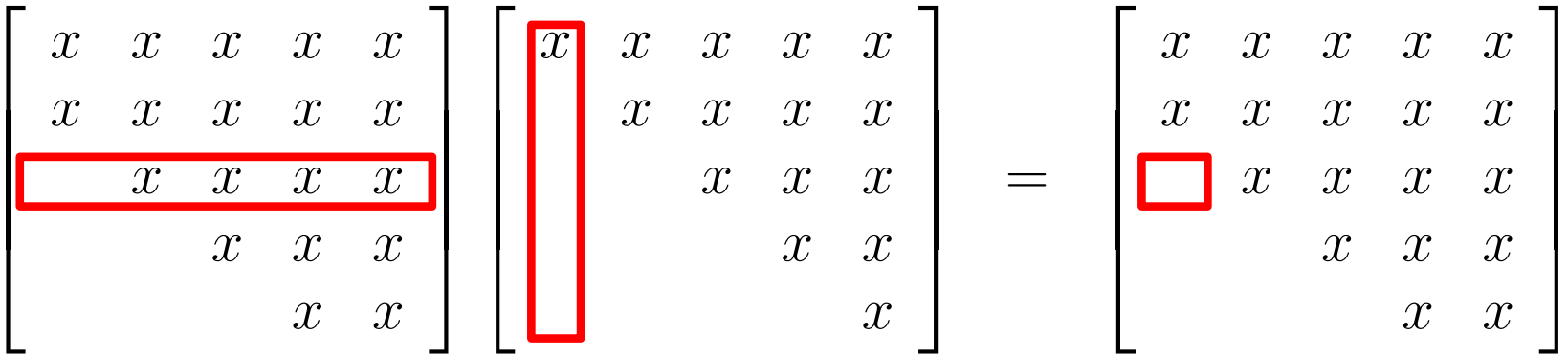
  and $\mathbf{Q}_k$ would also be upper Hessenberg.

- The reverse product, $\mathbf{R}_k\mathbf{Q}_k$, could also be effected in $O(n^2)$ operations using the same Givens rotations on the columns of $\mathbf{R}_k$, resulting in upper Hessenberg $\mathbf{A}_k$

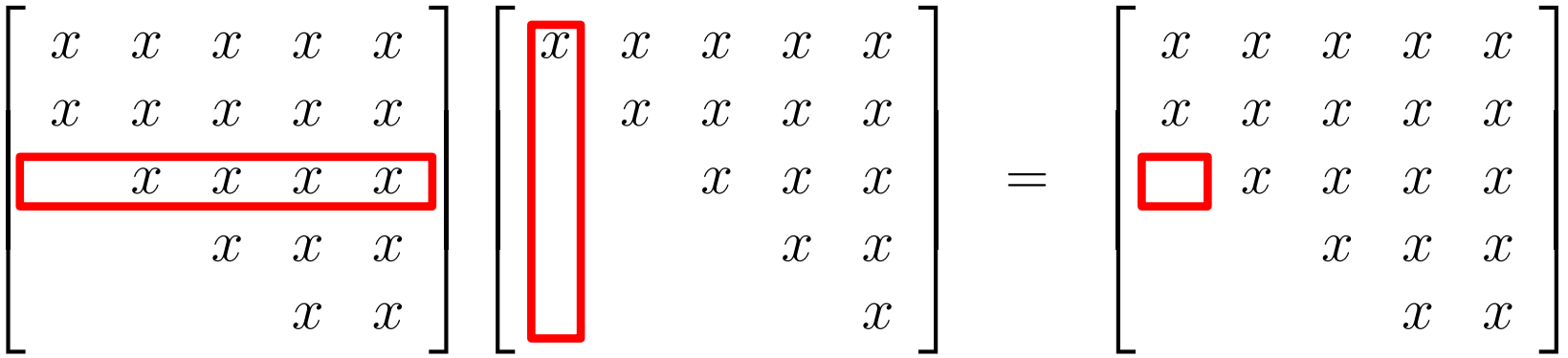
# Upper Hessenberg X Upper Triangular is Upper Hessenberg

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

# Upper Hessenberg X Upper Triangular is Upper Hessenberg



# Upper Hessenberg X Upper Triangular is Upper Hessenberg



- 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 $O(n^3)$.
- "Reverse" Givens can be used to compute $A_k = R_k Q_k$.
- 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

- 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 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

- QR iteration is implemented in two stages

$$\begin{array}{lcll} \text{symmetric} & \longrightarrow & \text{tridiagonal} & \longrightarrow \text{ diagonal} \\ & & \text{or} & \\ \text{general} & \longrightarrow & \text{Hessenberg} & \longrightarrow \text{ triangular} \end{array}$$

- Preliminary reduction requires definite number of steps whereas subsequent iterative stage continues until convergence
- Most efficient approach for preliminary reduction of general matrix to uupper Hessenberg is to use Householder transformations, requiring $O(n^3)$ operations
- With $A_0$ in upper Hessenberg form, only modest number of iterations usually required in practice so much of the work is in the preliminary reduction unless eigenvectors are needed
- Cost of accumulating eigenvectors, if needed, dominates total cost
- **Note:** it is important that initial reduction is only to *upper Hessenberg* and not upper triangular.

# Cost of QR Iteration

Approximate overall cost of preliminary iteration 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

# Stopped Here

# Preliminary Reduction to Upper Hessenberg

- Cost per QR iteration is $O(n^2)$ if $\mathbf{A}_0$ is upper Hessenberg

- The following sequence of Householder reflections will produce $\mathbf{A}_0$ in $O(n^3)$ operations, starting with $\mathbf{Q} = \mathbf{I}$, $\mathbf{A}_0 = \mathbf{A}$,

$$
\begin{array}{l}
\textbf{for } k = 1 : n-2 \\
\quad \mathbf{a} = 0 \\
\quad a(k+1:n) = \mathbf{A}_0(k+1:n, k) \\
\quad \alpha = -\|\mathbf{a}\|_2 \operatorname{sign}(a_{k+1}) \\
\quad \mathbf{v} = \mathbf{a} - \alpha \mathbf{e}_{k+1} \\
\quad \left.\begin{array}{l} \mathbf{H} = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}} \\ \mathbf{A}_0 = \mathbf{H}\mathbf{A}_0\mathbf{H}^T \\ \mathbf{Q} = \mathbf{Q}\mathbf{H}^T \end{array}\right\} \begin{array}{l} \text{Perform Householder update} \\ \mathbf{H} = \mathbf{H}^T = \mathbf{H}^{-1} \end{array} \\
\textbf{end}
\end{array}
$$

- Net result is $\mathbf{A}_0 = \mathbf{Q}^T\mathbf{A}\mathbf{Q}$ is upper Hessenberg

# Preliminary Reduction to Upper Hessenberg

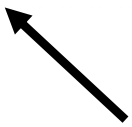
*prelim.m demo…*

```
hdr;

n=8;

I=eye(n);

ifnew_A=1;
if ifnew_A > 0;
  A=rand(n,n);
  [V,D]=eig(A); d=diag(D); [dmx,imx]=max(abs(d));
  D(imx,imx)=D(imx,imx)/dmx;
  A=real(V*D*inv(V));
end;

%% Preliminary Reduction

A0=A; Q=I;
for k=1:n-2;
    a=zeros(n,1); a(k+1:n)=A0(k+1:n,k); alpha=norm(a);
    if a(k+1) > 0; alpha=-alpha; end;
    v=a-alpha*I(:,k+1); s=2/(v'*v);
    H  = I-s*v*v';
    A0 = H*A0*H;
    Q  = Q*H;

    if n<9; A0(abs(A0)<10*eps)=0; A0, end;

end;
Q2=Q;

if n < 9; A0-Q2'*A*Q2,  end;  %% Test preliminary reduction
```

*Not efficient! Use factored form*

# Preliminary Reduction to Upper Hessenberg

```
A0 =
  -0.0435  -0.2968   0.0178   0.4857   0.1602  -0.0476  -0.2258   0.0271
  -0.6717   0.4235   0.1185   0.0394   0.0867  -0.4936  -0.3725   0.0952
        0   0.4110   0.6130  -0.0399  -0.0282   0.3363  -0.0358   0.2663
        0        0  -0.5803   0.1457   0.1958  -0.1080   0.2731  -0.5637
        0        0        0  -0.2930   0.4563  -0.0495   0.3867  -0.2996
        0        0        0        0  -0.8980   0.0793   0.0184  -0.1702
        0        0        0        0        0  -0.3290   0.5557  -0.0854
        0        0        0        0        0         0   0.9233   0.0379

ans =
            0            0  -4.1633e-17   5.5511e-17   5.5511e-17  -3.4694e-17  -2.7756e-17   1.0408e-17
            0            0  -2.7756e-17            0  -5.5511e-17   1.6653e-16   1.6653e-16  -1.3878e-17
  -1.3751e-18   1.1102e-16   3.3307e-16   4.8572e-17  -7.6328e-17   1.6653e-16  -1.5959e-16   1.6653e-16
   8.0414e-17  -4.2486e-17            0  -1.1102e-16            0   6.9389e-17  -1.1102e-16            0
   4.7232e-17   2.7702e-17  -1.0838e-16            0  -5.5511e-17   1.5959e-16            0   5.5511e-17
   3.7402e-17   7.6098e-17   1.1446e-16  -3.5095e-18  -1.1102e-16  -2.7756e-17   9.7145e-17            0
  -4.3459e-17  -2.5097e-17  -3.6111e-17  -4.6563e-17  -1.3969e-16   5.5511e-17  -2.2204e-16   8.3267e-17
   9.1692e-17   1.3574e-17  -1.7567e-16  -4.4028e-17  -1.9164e-16  -6.0192e-17            0  -1.3878e-17
```



- $\mathbf{A}_0 - \mathbf{Q}^T\mathbf{A}\mathbf{Q}$

# Finding All Eigenpairs

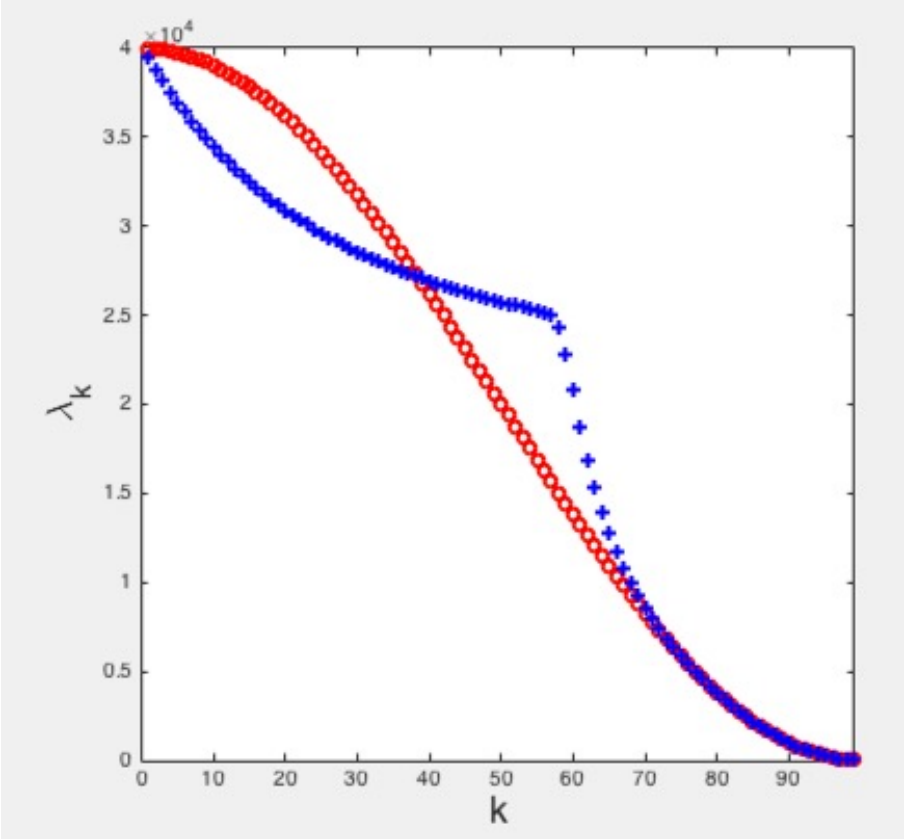
*sym_eig_qr_w_shift.m  demo…*

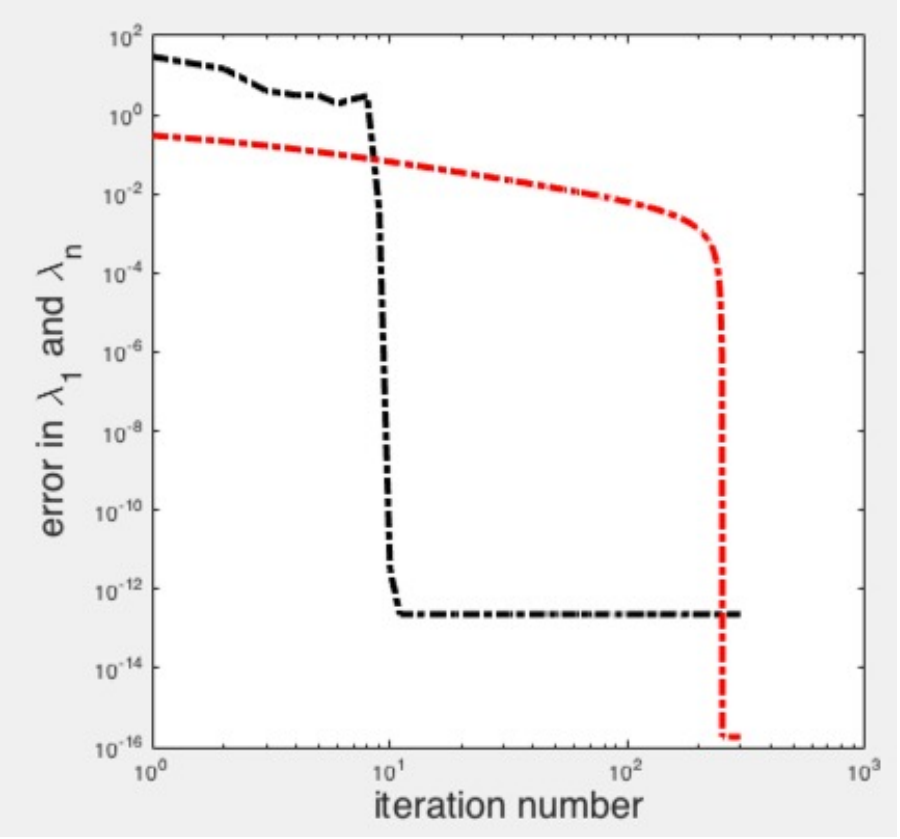
```
for k=1:300;

    %   QR w/ SHIFT
    I=eye(n);
    [Q,R]=qr(X-sigma*I);
    X=(R*Q)+sigma*I;

    d(1:n)=diag(X); lmin(k) = min(d); lmax(k)=max(d);

    %   DEFLATION
    n1=max(1,n-1); if abs(X(n,n1)) < eps*eps; X=X(1:n1,1:n1); n=n1; [k n], end;

    kk(k)=k;
    plot(jj,lj,'ro',jj,d,'b+',lw,2); axis([0 no -1 lmx]);
    xlabel('k',fs,18); ylabel('\lambda_k',fs,18); axis square; pause(0.101);

    %   WILKINSON SHIFT
    n1=max(1,n-1);
    An = X(n1:n,n1:n); [V,D]=eig(full(An)); sigma=0;
    sigma=D(1,1); if abs(D(end,end)) < abs(sigma); sigma=D(end,end); end;
end;
```

# Finding All Eigenpairs

- Once λ_n has converged ($R_{nn} \rightarrow \lambda_n$),
  - Reduce matrix: n → n-1
  - Continue with shifted QR to target next smallest eigenvalue.

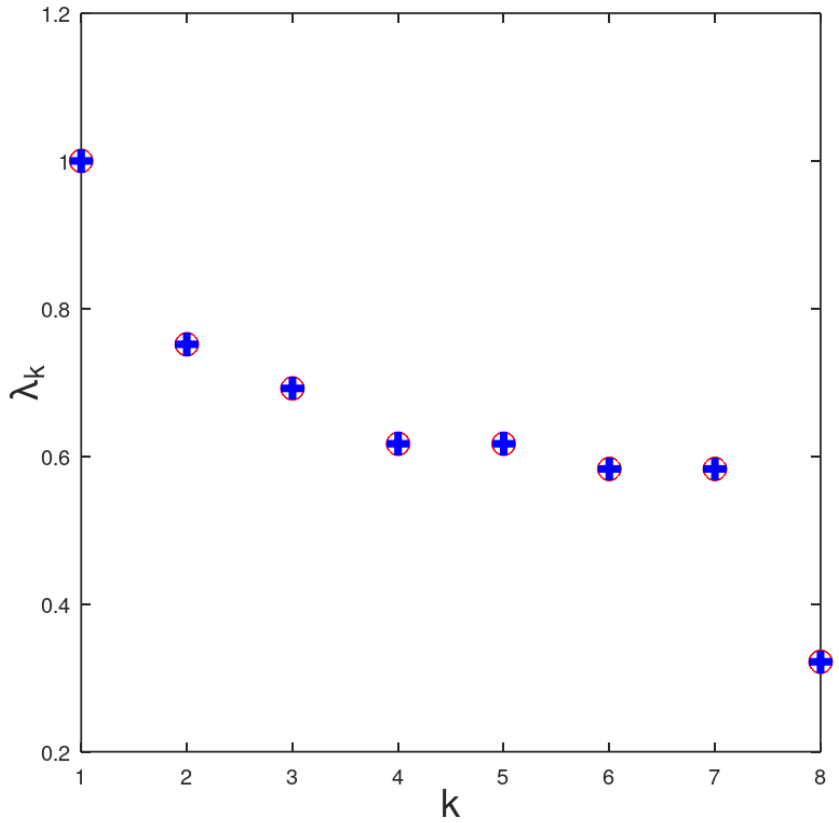
  - *sym_eig_qr_w_shift.m*
  - *full_qr_iter.m*



```
% sym_eig_qr_w_shift.m

hdr;

n = 80;
h = 1/(n+1);    x=1:n; x=h*x';
e = ones(n,1); A = spdiags([-e 2*e -e],-1:1, n,n)/(h*h);
jj=1:n; lj= 2*(1-cos(jj*pi*h))/(h*h); lj=lj(n:-1:1);

rng('default'); seed=1; rng(seed);
z=rand(n,1);

X=A; sigma=0;

no=n; lmx = 4*(n+1)^2;
d=lj;

  n0=n; kl=0; kn=zeros(n,1);
  for k=1:3*n0;

      %   QR w/ SHIFT
      I=eye(n);
      [Q,R]=qr(X-sigma*I);
      X=(R*Q)+sigma*I;

      d(1:n)=diag(X); lmin(k) = min(d); lmax(k)=max(d); kk(k)=k;

      %   DEFLATION
      n1=max(1,n-1);
      if abs(X(n,n1))<eps*eps; X=X(1:n1,1:n1);
         kn(n)=k-kl; kl=k; disp([n k kn(n)])
         n=n1; if n==1; break; end;
      end;

      plot(jj,lj,'ro',jj,d,'b+',lw,2); axis([0 no -1 lmx]);
      xlabel('k',fs,18); ylabel('\lambda_k',fs,18); axis square; pause(0.011);

      %   WILKINSON SHIFT
      n1=max(1,n-1);
      An = X(n1:n,n1:n); [V,D]=eig(full(An)); sigma=0;
      sigma=D(1,1); if abs(D(end,end)) < abs(sigma); sigma=D(end,end); end;
  end;

  emin = 2*(1-cos(pi*h))/(h*h); emax = lj(1);
  errn = abs(emin-lmin)/emin;
  errx = abs(emax-lmax)/emax;

  figure
  semilogy(kk,errn,'k-.',kk,errx,'r-.',lw,2)
  ylabel('error in \lambda_1 and \lambda_n',fs,18);
  xlabel('iteration number',fs,18); axis square;
  legend('Convergence of smallest eigenpair',...
         'Convergence of largest eigenpair','location','northeast')
```

```
hdr;  %% full_qr_iter.m

if ifnew_A > 0;
  n=8;
  A=rand(n,n);
  [V,D]=eig(A); d=diag(D); [dmax,imax]=max(abs(d)); D(imax,imax)=D(imax,imax)/dmax;
  A=real(V*D*inv(V));
  A0=A;
end;

%% Preliminary Reduction
I=eye(n); A=A0; Q=I;
for k=1:n-2;
    a=zeros(n,1); a(k+1:n)=A(k+1:n,k); alpha=norm(a);
    if a(k+1) > 0; alpha=-alpha; end;
    v=a-alpha*I(:,k+1); s=2/(v'*v);
    H = I-s*v*v';
    A = H*A*H;
    Q = Q*H;
end;
Q1=Q; R1=A;

A=A0;   %% Use original square A
A=R1;   %% Use upper Hessenberg

lj=eig(A); d=lj; [plj,ind]=sort(abs(lj),'descend');

no=n; X=A; sigma=0;
emax = max(abs(lj)); emin = min(abs(lj)); jj = [1:n]';

  n0=n; kl=0; kn=zeros(n,1);
  for k=1:10*n0;

      I=eye(n);               %   QR w/ SHIFT
      [Q,R]=qr(X-sigma*I);
      X=(R*Q)+sigma*I;

      d(1:n)=diag(X); lmin(k) = min(d); lmax(k)=max(d);

      %   DEFLATION
      n1=max(1,n-1);
      if abs(X(n,n1))<eps*eps; X=X(1:n1,1:n1);
         kn(n)=k-kl; kl=k; disp([n k kn(n)])
         n=n1; if n==1; break; end;
      end;

      kk(k)=k;
      pld=sort(abs(d),'descend');
      plot(jj,plj,'ro',jj,pld,'b+',lw,2); % axis([0 no -1 emax]);
      xlabel('k',fs,18); ylabel('\lambda_k',fs,18); axis square; pause(0.051);

      %   WILKINSON SHIFT
      n1=max(1,n-1);
      An = X(n1:n,n1:n); [V,D]=eig(full(An)); sigma=0;
      sigma=D(1,1); if abs(D(end,end)) < abs(sigma); sigma=D(end,end); end;
  end;
```



# 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 $\lambda_1$:

**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 $\mathbf{y} \neq 0$),

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

**Estimated eigenvalue**

**Actual eigenvalue**

- 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, $\mu_1$ is the maximum eigenvalue of $T_{k+1} := V_{k+1}^T A \, V_{k+1}$.

- *Notice that this is a similarity transformation for* $\mathbf{A}$ *if* $\mathbf{V}_{k+1}$ *is square (i.e., k+1=n)*

# 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 $\lambda_1$ is given by the eigenvalue $\mu_1$ of the much smaller $(k+1) \times (k+1)$ matrix, $T_{k+1}$.
- It is the closest approximation to $\lambda_1$ out of all possible polynomials of degree $k$ in $A$ and therefore superior (or equal to) the power method.
- Similarly, $\mu_k$ is the closest approximation to $\lambda_n$.
- The methods produce the best possible approximations (in this subspace) to the extreme eigenvalue/vector pairs.

# 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

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

- $\mathbf{x}_1$ is known as the Ritz vector and $\mu_1$ the Ritz value

# 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 algorithm 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 $\mathcal{K}_k$.

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

$$Q_k := (\mathbf{q}_1\ \mathbf{q}_2\ \cdots\ \mathbf{q}_k)$$

- 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,

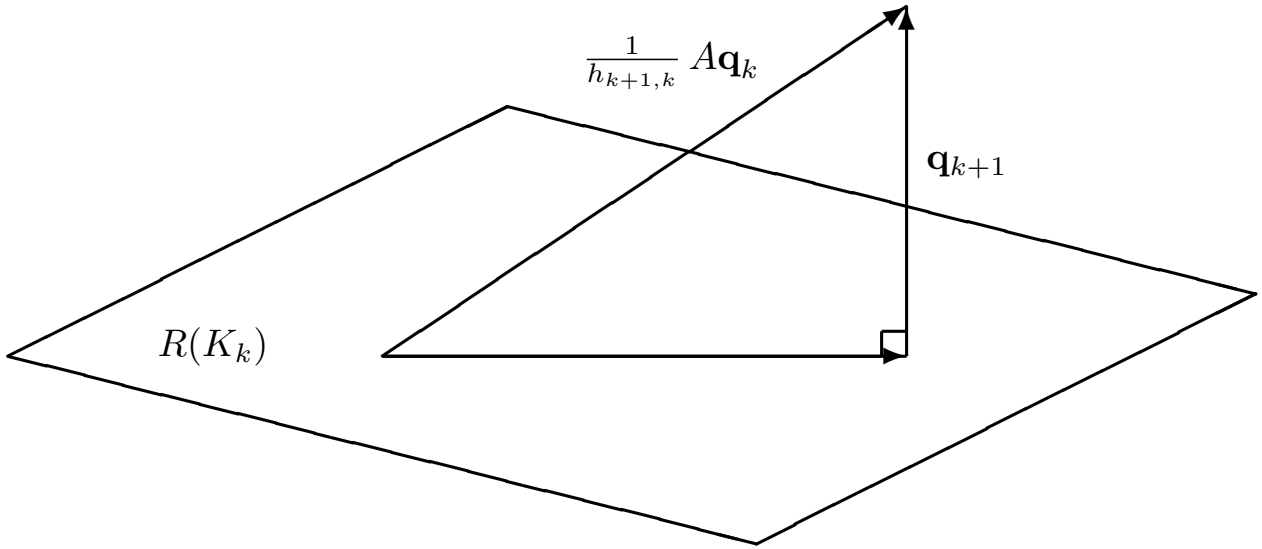
$$A\,[\mathbf{q}_1 \;\; \mathbf{q}_2 \;\; \cdots \;\; \mathbf{q}_k] \;=\; [\mathbf{q}_1 \;\; \mathbf{q}_2 \;\; \cdots \;\; \mathbf{q}_k] \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \beta_2 & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{bmatrix} \;+\; \beta_k \mathbf{q}_{k+1}\mathbf{e}_k^T .$$

# 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 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 Methods

- Krylov subspace methods reduce matrix to Hessenberg (or tridiagonal) form using only matrix-vector multiplication

- For arbitrary starting vector $\mathbf{x}_0$, if

$$\mathbf{K}_k = [\mathbf{x}_0 \ \mathbf{A}\mathbf{x}_0 \ \cdots \ \mathbf{A}^{k-1}\mathbf{x}_0]$$

  then

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

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

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

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

  so that

$$\mathbf{Q}_n^H\mathbf{A}\mathbf{Q}_n \ = \ \mathbf{R}_n\mathbf{C}_n\mathbf{R}_n^{-1} \ =: \ \mathbf{H}$$

  with $\mathbf{H}$ upper Hessenberg

# Krylov Subspace Methods, continued

- For each $k = 1, \ldots, n$, define the $n \times k$ Krylov matrix

$$\mathbf{K}_k = \begin{bmatrix} \mathbf{x}_0 & \mathbf{x}_1 & \cdots & \mathbf{x}_{k-1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_0 & \mathbf{A}\mathbf{x}_0 & \cdots & \mathbf{A}^{k-1}\mathbf{x}_0 \end{bmatrix},$$

  and the corresponding Krylov subspace to be the column space

$$\mathcal{K}_k = \mathcal{R}(\mathbf{K}_k)$$

- For $k = n$,

$$\begin{aligned} \mathbf{A}\mathbf{K}_n &= \begin{bmatrix} \mathbf{A}\mathbf{x}_0 & \cdots & \mathbf{A}\mathbf{x}_{n-2} & \mathbf{A}\mathbf{x}_{n-1} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_{n-1} & \mathbf{x}_n \end{bmatrix} \\ &= \mathbf{K}_n \begin{bmatrix} \mathbf{e}_2 & \cdots & \mathbf{e}_n & \mathbf{a} \end{bmatrix} =: \mathbf{K}_n \mathbf{C}_n \end{aligned}$$

  with $\mathbf{a} = \mathbf{K}_n^{-1}\mathbf{x}_n$

# 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.

$$H \;=\; \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \end{bmatrix}, \quad h_{ij} \;=\; 0 \text{ for } i > j+1.$$

- Proof:

$$\begin{aligned} \text{If} \quad \mathbf{v}_j &\in \mathit{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_j\} \;\equiv\; \mathbb{P}_{j-1}(A)\mathbf{x} \;\equiv \mathcal{K}_j \\ \text{then} \quad A\mathbf{v}_j &\in \mathit{span}\{A\mathbf{q}_1, \ldots, A\mathbf{q}_j\} \;\subset\; \mathbb{P}_j(A)\mathbf{x} \;\equiv \mathcal{K}_{j+1}. \\ \mathbf{q}_i^T A\mathbf{v}_j &= 0, \quad i > j+1, \end{aligned}$$

because, for $i > j+1$, $\mathbf{q}_i \perp \mathcal{K}_{j+1}$.

# Relationship of K and C

- Start with the Krylov matrix $\mathbf{K} = \begin{bmatrix} \mathbf{x} & \mathbf{Ax} & \mathbf{A}^2\mathbf{x} & \cdots & \mathbf{A}^{k-1}\mathbf{x} \end{bmatrix}$

- Multiply by $\mathbf{A}$ to yield $\mathbf{AK} = \begin{bmatrix} \mathbf{Ax} & \mathbf{A}^2\mathbf{x} & \mathbf{A}^3\mathbf{x} & \cdots & \mathbf{A}^{k}\mathbf{x} \end{bmatrix}$

- Notice that the first $k-1$ columns of $\mathbf{AK}$ are just the second through $k$th columns of $\mathbf{K}$:

$$\mathbf{Ax} = \text{col } 2 \quad \mathbf{A}^2\mathbf{x} = \text{col } 3 \ldots, \ \mathbf{A}^{k-1}\mathbf{x} = \text{col } k$$

- The remaining term, $\mathbf{A}^k\mathbf{x}$, is to be determined

- Since the vectors $\{\mathbf{x}, \mathbf{Ax}, \mathbf{A}^2\mathbf{x}, \ldots, \mathbf{A}^{k-1}\mathbf{x}\}$ are assumed to form a basis (or at least to be linearly independent in the Krylov subspace), the vector $\mathbf{A}^k\mathbf{x}$ can be expressed as a linear combination of them:

$$\mathbf{A}^k\mathbf{x} = c_0\mathbf{x} + c_1\mathbf{Ax} + c_2\mathbf{A}^2\mathbf{x} + \cdots + c_{k-1}\mathbf{A}^{k-1}\mathbf{x}$$

- This relation represents the characteristic polynomial for $\mathbf{A}$ relative to the vector $\mathbf{x}$:

$$p_k(\mathbf{A})\mathbf{x} := \mathbf{A}^k\mathbf{x} - c_{k-1}\mathbf{A}^{k-1}\mathbf{x} - \cdots - c_1\mathbf{A}\mathbf{x} - c_0\mathbf{x} = \mathbf{0}$$

- Define the companion matrix $\mathbf{C}$ by

$$\mathbf{C} = \begin{pmatrix} 0 & 0 & \cdots & 0 & c_0 \\ 1 & 0 & \cdots & 0 & c_1 \\ 0 & 1 & \cdots & 0 & c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & c_{k-1} \end{pmatrix}$$

- The effect of post-multiplying $\mathbf{K}$ by $\mathbf{C}$ is to "shift" the columns of $\mathbf{K}$ one position to the left, and the last column of $\mathbf{KC}$ becomes the linear combination of the columns of $\mathbf{K}$ with the coefficients $c_0, c_1, \ldots, c_{k-1}$.

- Explicitly,

$$\begin{aligned} \mathbf{KC} &= \begin{bmatrix} \mathbf{A}\mathbf{x} & \mathbf{A}^2\mathbf{x} & \cdots & \mathbf{A}^{k-1}\mathbf{x} & p_k(\mathbf{A})\mathbf{x} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}\mathbf{x} & \mathbf{A}^2\mathbf{x} & \cdots & \mathbf{A}^{k-1}\mathbf{x} & \mathbf{A}^k\mathbf{x} \end{bmatrix} = \mathbf{AK} \end{aligned}$$

- Thus, we obtain the matrix equality $\mathbf{AK} = \mathbf{KC}$

# Krylov Subspace Methods

- Krylov subspace methods reduce matrix to Hessenberg (or tridiagonal) form using only matrix-vector multiplication

- For arbitrary starting vector $\mathbf{x}_0$, if

$$\mathbf{K}_k = [\mathbf{x}_0 \ \mathbf{A}\mathbf{x}_0 \ \cdots \ \mathbf{A}^{k-1}\mathbf{x}_0]$$

  then

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

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

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

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

  so that

$$\mathbf{Q}_n^H\mathbf{A}\mathbf{Q}_n \ = \ \mathbf{R}_n\mathbf{C}_n\mathbf{R}_n^{-1} \ =: \ \mathbf{H}$$

  with $\mathbf{H}$ upper Hessenberg

- *As discussed earlier, QR factorization via Gram-Schmidt*

# Krylov Subspace Methods

- Equating $k$th columns on each side of $\mathbf{AQ}_n = \mathbf{Q}_n\mathbf{H}$ yields recurrence

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

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

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

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

$$\mathbf{Q}_k = [\mathbf{q}_1 \cdots \mathbf{q}_k]$$

  then

$$\mathbf{H}_k = \mathbf{Q}_k^H \mathbf{A} \mathbf{Q}_k$$

  is an upper Hessenberg matrix

- Eigenvalues of $\mathbf{H}_k$, called *Ritz values*, are approximate eigenvalues of $\mathbf{A}$

- *Ritz vectors*, given by $\mathbf{Q}_k\mathbf{y}$ are approximate eigenvectors of $\mathbf{A}$, where $\mathbf{y}$ is an eigenvector of $\mathbf{H}_k$

- Eigenpairs of $\mathbf{H}_k$ must be computed by another method, such as QR iteration, but this is an easier problem if $k \ll n$

# Arnoldi Iteration, continued

- Arnoldi iteration is fairly expensive in work and storage because each new vector $\mathbf{q}_k$ must be orthogonalized against all previous columns of $\mathbf{Q}_k$ and all must be stored for that purpose

- So Arnoldi process is usually restarted periodically with carefully chosen starting vector (e.g., current approximate eigenvector)

- Ritz values and vectors produced are often good approximations to eigenvalues and eigenvectors of $\mathbf{A}$ after relatively few iterations

# 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,

$$A\,[\mathbf{q}_1 \;\; \mathbf{q}_2 \;\; \cdots \;\; \mathbf{q}_k] \;=\; [\mathbf{q}_1 \;\; \mathbf{q}_2 \;\; \cdots \;\; \mathbf{q}_k] \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \beta_2 & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{bmatrix} \;+\; \beta_k \mathbf{q}_{k+1}\mathbf{e}_k^T .$$

# Lanczos Iteration

- $\alpha_k$ and $\beta_k$ are diagonal and subdiagonal entries of symmetric tridiagonal matrix $\mathbf{T}_k$

- As with Arnoldi, Lanczos iteration does not produce eigenvalues and eigenvectors directly, but only tridiagonal matrix $\mathbf{T}_k$, whose eigenpairs must be computed by another method to obtain the 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 $k = n$ the resulting tridiagonal matrix would be orthogonally similar to $\mathbf{A}$

- In practice, rounding error causes loss of orthogonality, invalidating this expectation

- Problem can be overcome by re-orthogonalizing 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

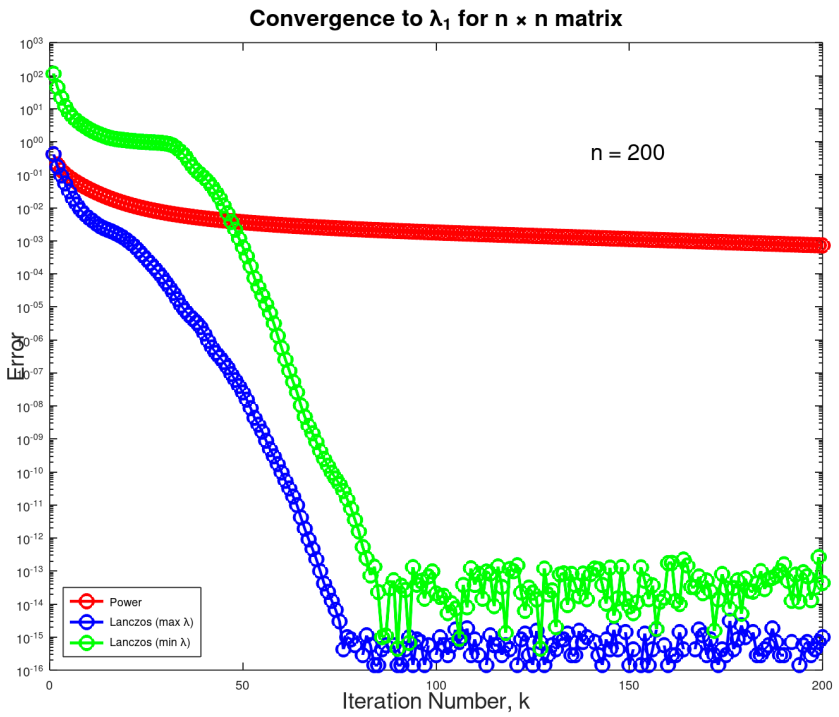
- Great virtue of Arnoldi/Lanczos iterations is their ability to produce good approximations of extreme eigenvalues for $k \ll n$

- Moreover, the require only one matrix-vector multiply by $\mathbf{A}$ per step and little auxialiary storage, so are ideally suited to large sparse matrices

- If eigenvalues are needed in middle of spectrum, say near $\sigma$, algorithm can be applied to $(\mathbf{A} - \sigma\mathbf{I})^{-1}$, assuming it is practical to solve systems of form $(\mathbf{A} - \sigma\mathbf{I})\mathbf{x} = \mathbf{y}$. (Presumably with a preconditioned iterative method.)

```
%%  Comparison of Lanczos and Power Iteration
clear all; close all; lw='linewidth'; fs='fontsize';

n=200;
D=diag(1:n);
A=rand(n,n); [Q,R]=qr(A); A=Q'*D*Q;  A=.5*(A+A');
lam1=n; lamn=1;

q1=rand(n,1); q1=q1/norm(q1); q0=0*q1; beta(1)=0;

x=q1;  % Initialize x for Power iteration

m=n;   % Max number of iterations

T=zeros(m,m);  % Normally, this is sparse, and tridiagonal.
kk=zeros(m,1);
for k=1:m;

   u=A*q1;                %% LANCZOS ITERATION
   alpha(k) = q1'*u;
   u = u-beta(k)*q0-alpha(k)*q1;
   beta(k+1) = norm(u);
   q0=q1;
   q1=u/beta(k+1);
   T(k,k)=alpha(k); T(k,k+1)=beta(k+1);T(k+1,k)=beta(k+1);
   [V,D]=eig(T(1:k,1:k));
     lmax(k)=max(diag(D));
     lmin(k)=min(diag(D));

   y=A*x;                 %% POWER ITERATION
   est(k)=x'*y;
   kk (k)=k;
   x=y/norm(y);

end;

hold off; semilogy(kk,abs(est-lam1)/lam1,'ro-',lw,1.3);
title('Convergence to \lambda_1 for n \times n matrix',fs,18);
ylabel('Error',fs,18); xlabel('Iteration Number, k',fs,18);
s=['n = ' num2str(n)]; text(.7*n,0.5,s,fs,18);
pause(1); hold on
emx=abs(lmax-lam1)/lam1; semilogy(kk,emx,'bo-',lw,1.3); pause(1)
emn=abs(lmin-lamn)/lamn; semilogy(kk,emn,'go-',lw,1.3);
legend('Power','Lanczos (max \lambda)',...
       'Lanczos (min \lambda)','location','southwest')
```

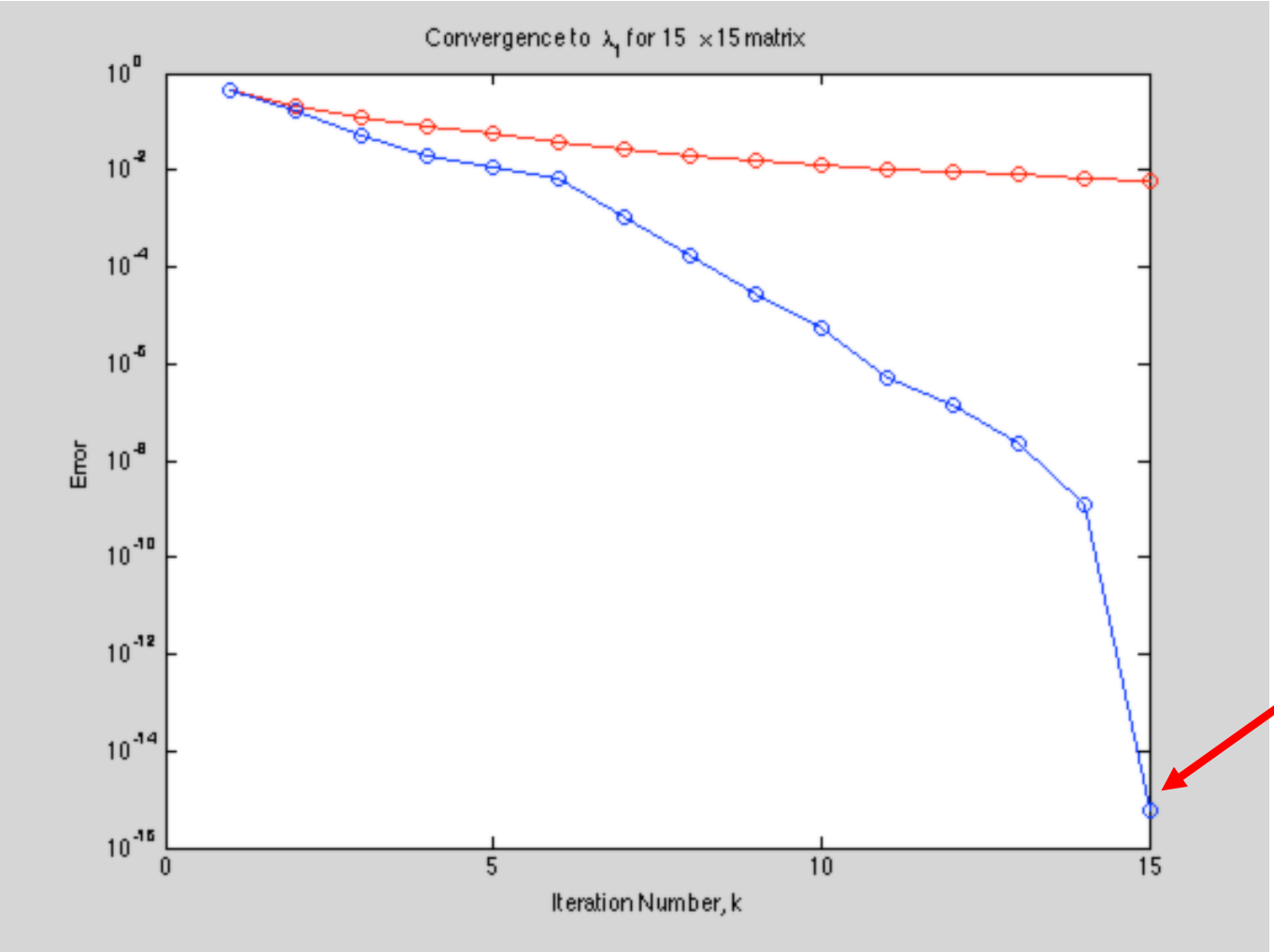
## *lanczos_and_power.m  demo…*




- *Lanczos max eigenpair converges much more rapidly than power iteration does to $\lambda_{max}$*
- *Even convergence to $\lambda_{min}$ is relatively fast*

# Matlab Demo

- Lanczos vs Power Iteration. *(lanczos_and_power.m)*




*Q: What is happening here?*

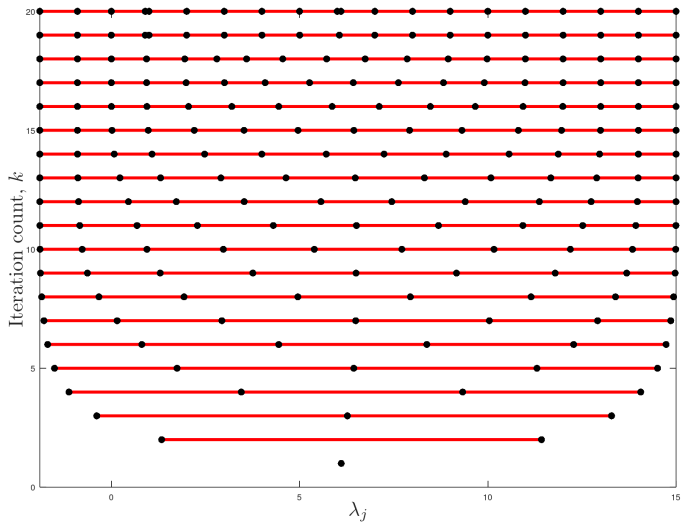
- Lanczos does a reasonable job of converging to extreme eigenvalues.

# *Convergence to All Eigenvalues*

*lanczos_plot_eigs.m  demo*

- Although we would generally not use Lanczos for all eigenpairs, it is interesting to do so to understand how Lanczos converges.
- Typically, the extreme eigenpairs emerge first
- Here, we look at a 29x29 example (see text) and plot the $k$ eigenvalue estimates after the $k$th iteration.

```
%%  Lanczos CONVERGENCE HISTORY
hdr; hold off;

D=diag([ .1 1.1  2 3 4 5 6 7 8 8.1 9 10 11 2.9 12 13 14 15 16 17]);

%% MATRIX SETUP
n=size(D,1); I=eye(n); D=D-2*I;   % Shift by "-2"
a=sin([1:n*n]'); A=reshape(a,n,n)+n*I;
[Q,R]=qr(A); A=Q'*D*Q;
xm=min(min(D)); XM=max(max(D));

m=n;                          % Max number of iterations
kk=zeros(m,1); T=zeros(m,m);  % Normally, this is sparse, and tridiagonal.
q1=ones(n,1); q1=q1/norm(q1); q0=0*q1; beta(1)=0;
for k=1:m;

   u=A*q1;     %  LANCZOS ITERATION
   alpha(k) = q1'*u;
   u = u-beta(k)*q0-alpha(k)*q1;
   beta(k+1) = norm(u);
   q0=q1;
   q1=u/beta(k+1);
   T(k,k)=alpha(k); T(k,k+1)=beta(k+1);T(k+1,k)=beta(k+1);
   [V,D]=eig(T(1:k,1:k)); d=diag(D); max(d); lmax(k)=max(d);

   plot(d,k+0*d,'r-',lw,2,d,k+0*d,'k.',ms,14); hold on;
   xlabel('$\lambda_j$',intp,ltx,fs,24);
   ylabel('Iteration count, $k$',intp,ltx,fs,24);
   axis([xm XM 0 m]); pause(.2)

end;
```




- *Lanczos extreme eigenpairs converge first*
- *Middle ones converge only at the end*

# 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, $k \ll n$.)

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

- Therefore, $\mu_1$ 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 $A$.

# Lanczos/Arnoldi Example Applications

1. Solving $A\mathbf{x} = \mathbf{b}$: Lanczos $\Longleftrightarrow$ conjugate gradient (CG) iteration ($A$-SPD)

   - Preconditioned CG: $A\mathbf{x} = \mathbf{b} \longrightarrow M^{-1}A\mathbf{x} = M^{-1}\mathbf{b} \Longleftrightarrow A\mathbf{x} = \lambda M\mathbf{z}$
   - PCG (Lanczos) yields generalized eigenpair estimates $(\lambda_k, \mathbf{z}_k)$ with no extra matvecs.
   - GMRES is the nonsymmetric Arnoldi equivalent.
   - Both CG and GMRES produce the *closest approximation* to $\mathbf{x}$ in $\mathcal{K}_k = \mathcal{R}(Q_k)$, in the appropriate norm:
     - $A$-norm for CG, with $A$ SPD
     - $A^TA$-norm for GMRES, with $A$ invertible.

2. Solving ordinary differential equations (ODEs)

$$\frac{du}{dt} = \lambda u, \quad u(t=0) = u^0 \implies u(t) = u^0 e^{\lambda t}$$

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u}, \quad \mathbf{u}(t=0) = \mathbf{u}^0 \implies \mathbf{u}(t) = e^{At}\mathbf{u}^0.$$

- Consider a small interval $\Delta t$: $\quad t^n := n\Delta t, \quad \mathbf{u}^n \approx \mathbf{u}(t^n)$.

  $e^{\lambda dt} \approx 1 + \lambda \Delta t \quad$ (Euler forward)

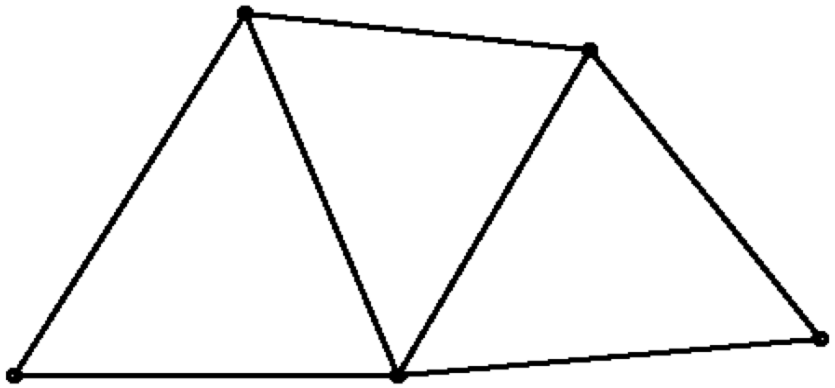
  $\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t A \mathbf{u}^n \approx e^{A\Delta t}\mathbf{u}^n.$

- Higher-order polynomials (and *rational* polynomials involving $I - \Delta t A$) are also possbile.

- For linear problems, can also use Lanczos/Anoldi:

$$\mathbf{u}^{n+1} = Q_k S_k e^{\mathcal{M}\Delta t} S_k^{-1} Q_k^T \mathbf{u}^n.$$

- *Stable* and *kth-order accurate.*



3. Graph Partitioning

- Lanczos can be used for partitioning large sparse graphs.
- Find $\mathbf{x}_{n-1}$, the eigenvector associated with the first nonzero eigenvalue of the Graph Laplacian, $G$.
  - $G_{ij} = -1$ if vertex $i$ connected to vertex $j$.
  - $G_{ii} =$ number of connections to vertex $i$.
  - $G_{ij} = 0$ otherwise.

- Upon sorting entries of $\mathbf{x}_2$ (the *Fiedler vector*), adjacent entries in the sorted rank are connected.
- Partitioning the sorted list into two parts gives two connected subgraphs.
- Recur, to get a set of connected subgraphs. (*Recursive spectral bisection.*)