Eigenvalue Problems and Iterative Methods

Motivation: Eigenvalue Problems

A matrix $A \in \mathbb{C}^{n \times n}$ has eigenpairs $(\lambda_1, v_1), \ldots, (\lambda_n, v_n) \in \mathbb{C} \times \mathbb{C}^n$ such that

$$Av_i = \lambda_i v_i, \quad i = 1, 2, \ldots, n$$

(We will order the eigenvalues from smallest to largest, so that $|\lambda_1| \leq |\lambda_2| \leq \cdots \leq |\lambda_n|$)

It is remarkable how important eigenvalues and eigenvectors are in science and engineering!

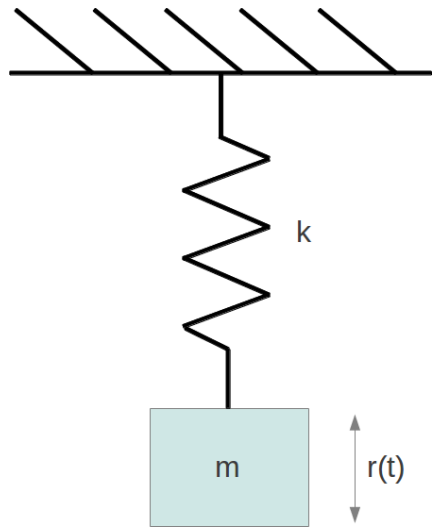
Motivation: Eigenvalue Problems

For example, eigenvalue problems are closely related to resonance

- Pendulums
- Natural vibration modes of structures
- Musical instruments
- Lasers
- Nuclear Magnetic Resonance (NMR)
- ...

Motivation: Resonance

Consider a spring connected to a mass



Suppose that:

- the spring satisfies Hooke's Law,[1] $F(t) = ky(t)$
- a periodic forcing, $r(t)$, is applied to the mass

[1] Here $y(t)$ denotes the position of the mass at time t

Motivation: Resonance

Then Newton's Second Law gives the ODE

$$y''(t) + \left(\frac{k}{m}\right) y(t) = r(t)$$

where $r(t) = F_0 \cos(\omega t)$

Recall that the solution of this non-homogeneous ODE is the sum of a homogeneous solution, $y_h(t)$, and a particular solution, $y_p(t)$

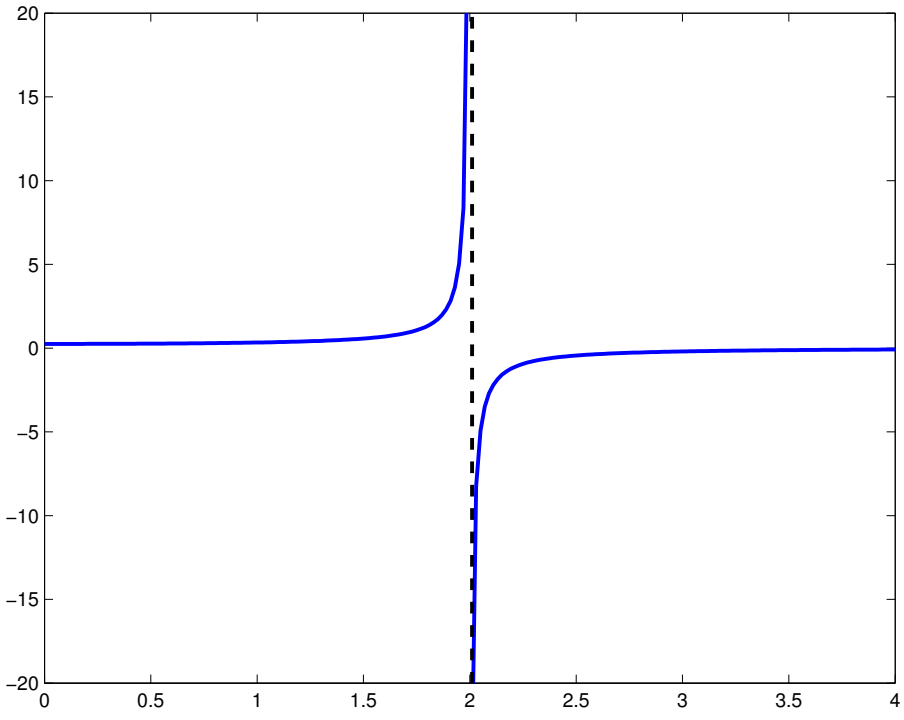
Let $\omega_0 \equiv \sqrt{k/m}$, then for $\omega \neq \omega_0$ we get:[2]

$$y(t) = y_h(t) + y_p(t) = C\cos(\omega_0 t - \delta) + \frac{F_0}{m(\omega_0^2 - \omega^2)}\cos(\omega t),$$

[2] C and δ are determined by the ODE initial condition

Motivation: Resonance

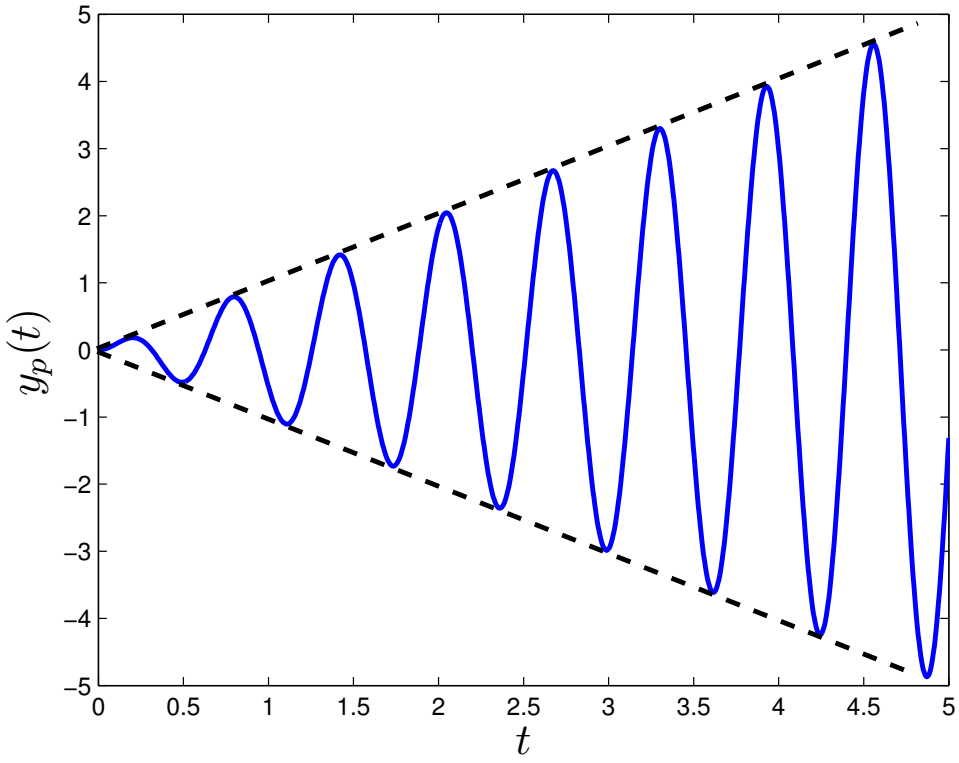
The amplitude of $y_p(t)$, $\frac{F_0}{m(\omega_0^2-\omega^2)}$, as a function of ω is shown below



Clearly we observe singular behavior when the system is forced at its natural frequency, i.e. when $\omega = \omega_0$

Motivation: Forced Oscillations

Solving the ODE for $\omega = \omega_0$ gives $y_p(t) = \frac{F_0}{2m\omega_0} t \sin(\omega_0 t)$



This is resonance!

Motivation: Resonance

In general, ω_0 is the frequency at which the unforced system has a non-zero oscillatory solution

For the single spring-mass system we substitute[3] the oscillatory solution $y(t) \equiv xe^{i\omega_0 t}$ into $y''(t) + \left(\frac{k}{m}\right) y(t) = 0$

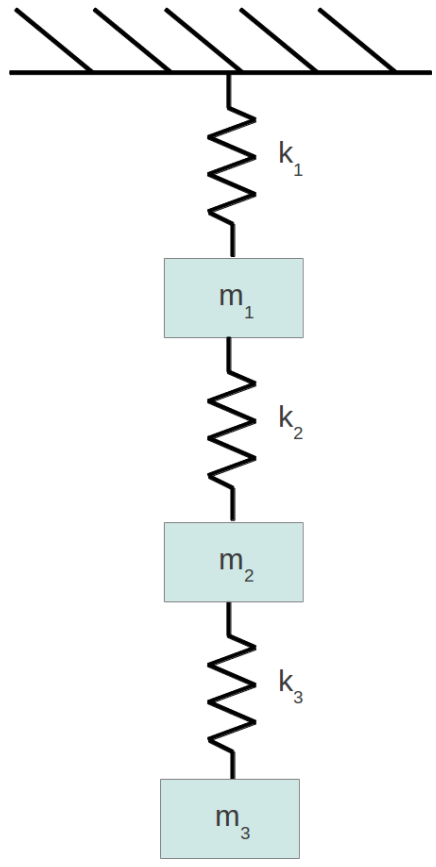
This gives a scalar equation for ω_0:

$$kx = \omega_0^2 mx \implies \omega_0 = \sqrt{k/m}$$

[3] Here x is the amplitude

Motivation: Resonance

Suppose now we have a spring-mass system with three masses and three springs



Motivation: Resonance

In the unforced case, this system is governed by the ODE system

$$My''(t) + Ky(t) = 0,$$

where M is the mass matrix and K is the stiffness matrix

$$M \equiv \begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix}, \quad K \equiv \begin{bmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix}$$

We again seek a nonzero oscillatory solution to this ODE, i.e. set $y(t) = xe^{i\omega t}$, where now $y(t) \in \mathbb{R}^3$

This gives the algebraic equation

$$Kx = \omega^2 Mx$$

Motivation: Eigenvalue Problems

Setting $A \equiv M^{-1}K$ and $\lambda = \omega^2$, this gives the eigenvalue problem

$$Ax = \lambda x$$

Here $A \in \mathbb{R}^{3\times 3}$, hence we obtain natural frequencies from the three eigenvalues λ_1, λ_2, λ_3

Motivation: Eigenvalue Problems

The spring-mass systems we have examined so far contain discrete components

But the same ideas also apply to continuum models

For example, the wave equation models vibration of a string (1D) or a drum (2D)

$$\frac{\partial^2 u(x,t)}{\partial t^2} - c\Delta u(x,t) = 0$$

As before, we write $u(x,t) = \tilde{u}(x)e^{i\omega t}$, to obtain

$$-\Delta\tilde{u}(x) = \frac{\omega^2}{c}\tilde{u}(x)$$

which is a PDE eigenvalue problem

Motivation: Eigenvalue Problems

We can discretize the Laplacian operator with finite differences to obtain an algebraic eigenvalue problem

$$Av = \lambda v,$$

where

- the eigenvalue $\lambda = \omega^2/c$ gives a natural vibration frequency of the system
- the eigenvector (or eigenmode) v gives the corresponding vibration mode

Motivation: Eigenvalue Problems

We will use the Python (and Matlab) functions `eig` and `eigs` to solve eigenvalue problems

- `eig`: find all eigenvalues/eigenvectors of a dense matrix
- `eigs`: find a few eigenvalues/eigenvectors of a sparse matrix

We will examine the algorithms used by `eig` and `eigs` in this Unit

Motivation: Eigenvalue Problems

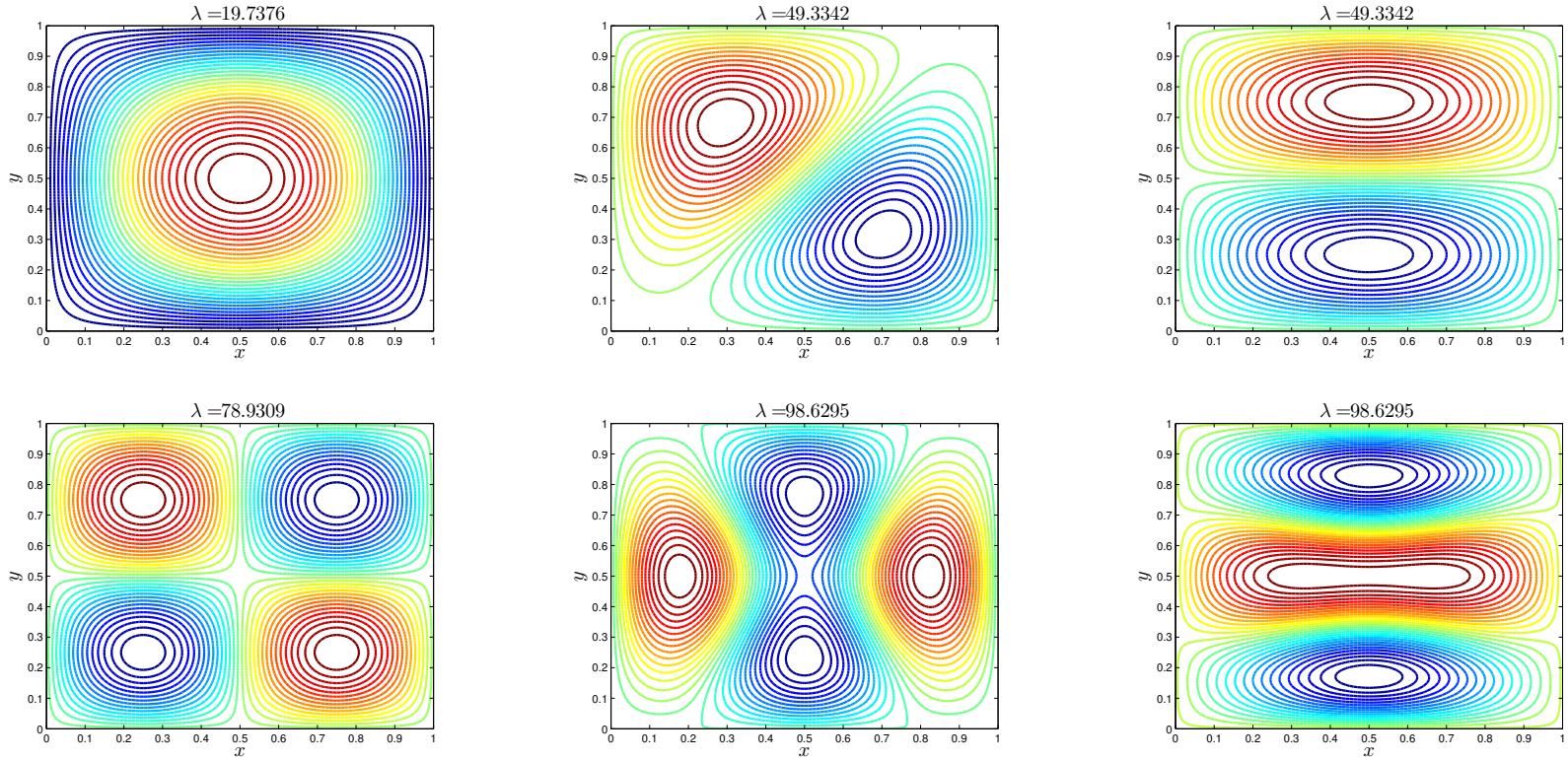
Python demo: Eigenvalues/eigenmodes of Laplacian on $[0,1]^2$, zero Dirichlet boundary conditions

Based on separation of variables, we know that eigenmodes are $\sin(\pi i x)\sin(\pi j y)$, $i,j = 1,2,\ldots$

Hence eigenvalues are $(i^2+j^2)\pi^2$

i	j	$\lambda_{i,j}$
1	1	$2\pi^2 \approx 19.74$
1	2	$5\pi^2 \approx 49.35$
2	1	$5\pi^2 \approx 49.35$
2	2	$8\pi^2 \approx 78.96$
1	3	$10\pi^2 \approx 98.97$
$\vdots$	$\vdots$	$\vdots$

Motivation: Eigenvalue Problems



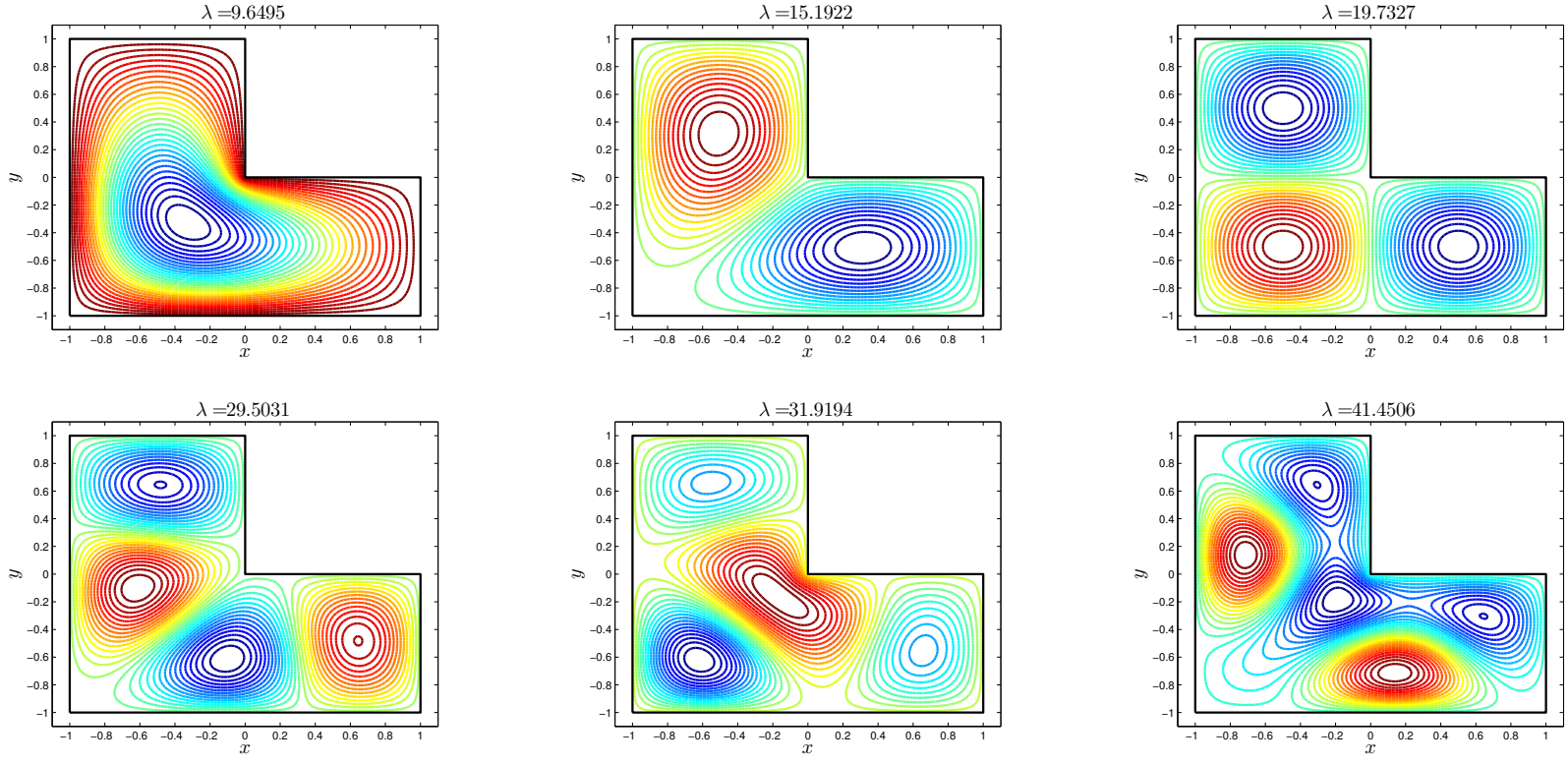
In general, for repeated eigenvalues, computed eigenmodes are L.I. members of the corresponding eigenspace

e.g. eigenmodes corresponding to $\lambda = 49.3$ are given by

$$\alpha_{1,2} \sin(\pi x) \sin(\pi 2y) + \alpha_{2,1} \sin(\pi 2x) \sin(\pi y), \quad \alpha_{1,2}, \alpha_{2,1} \in \mathbb{R}$$

Motivation: Eigenvalue Problems

And of course we can compute eigenmodes of other shapes...



Motivation: Eigenvalue Problems

A well-known mathematical question was posed by Mark Kac in 1966: "Can one hear the shape of a drum?"

The eigenvalues of a shape in 2D correspond to the resonant frequences that a drumhead of that shape would have

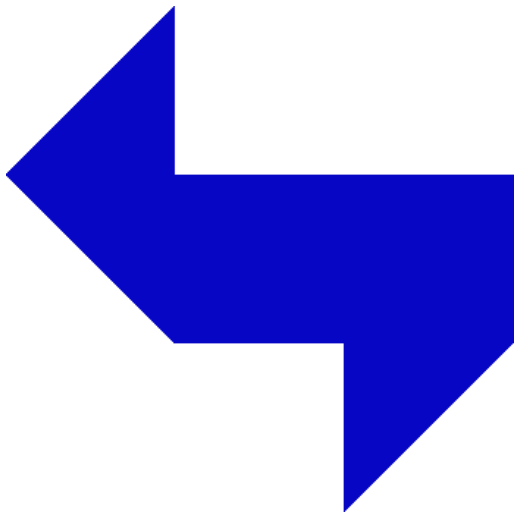
Therefore, the eigenvalues determine the harmonics, and hence the sound of the drum

So in mathematical terms, Kac's question was: If we know all of the eigenvalues, can we uniquely determine the shape?

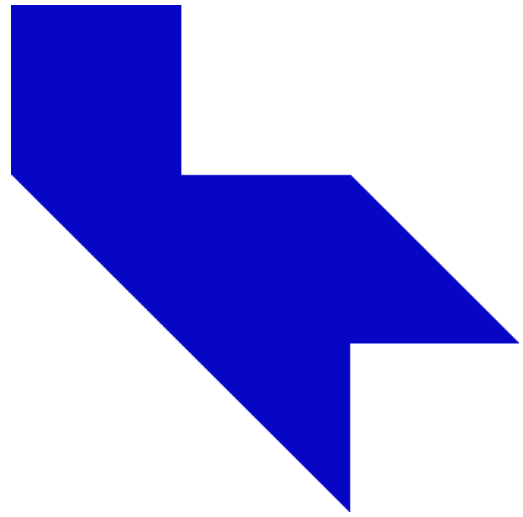
Motivation: Eigenvalue Problems

It turns out that the answer is no!

In 1992, Gordon, Webb, and Wolpert constructed two different 2D shapes that have exactly the same eigenvalues!



Drum 1



Drum 2

Motivation: Eigenvalue Problems

We can compute the eigenvalues and eigenmodes of the Laplacian on these two shapes using the algorithms from this Unit[4]

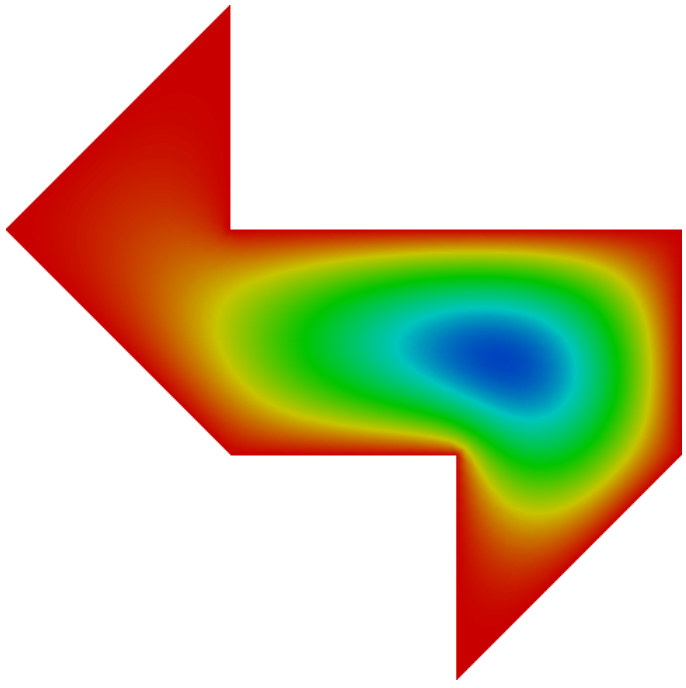
The first five eigenvalues are computed as:

	Drum 1	Drum 2
λ_1	2.54	2.54
λ_2	3.66	3.66
λ_3	5.18	5.18
λ_4	6.54	6.54
λ_5	7.26	7.26

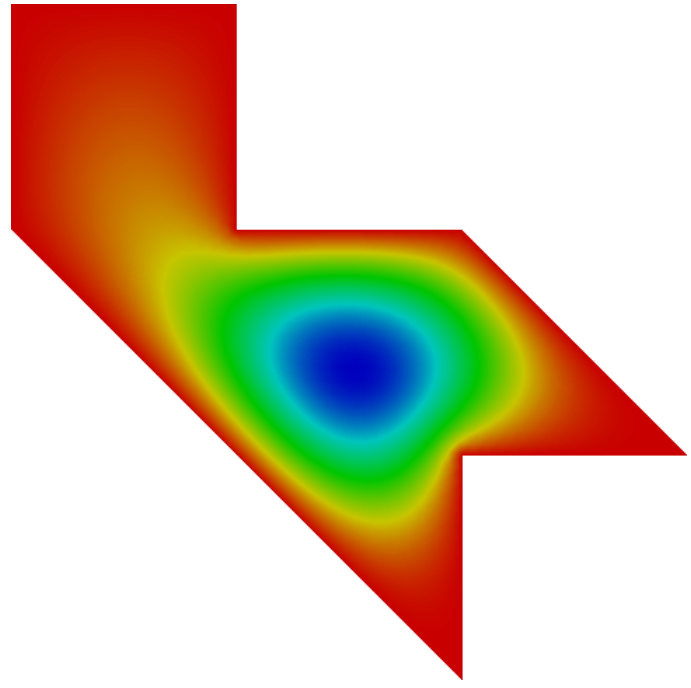
We next show the corresponding eigenmodes...

[4] Note here we employ the Finite Element Method (outside the scope of AM205), an alternative to F.D. that is well-suited to complicated domains

Motivation: Eigenvalue Problems

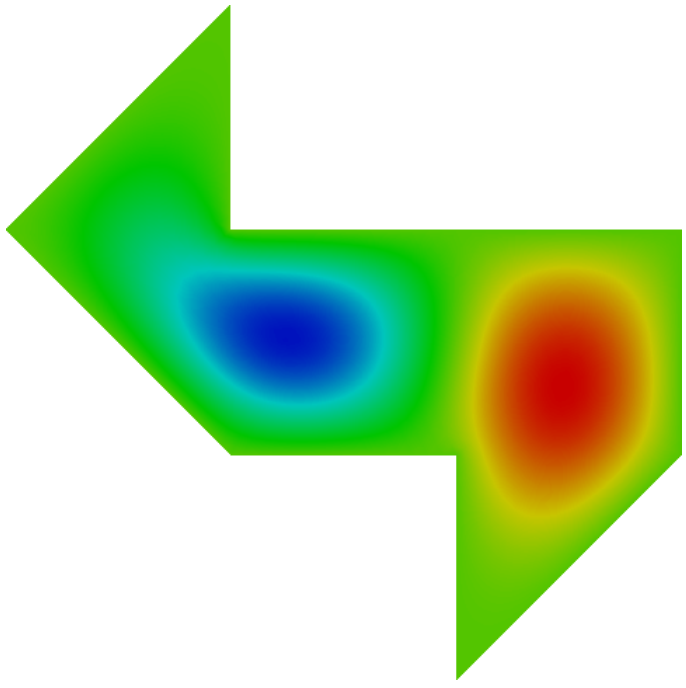


eigenmode 1

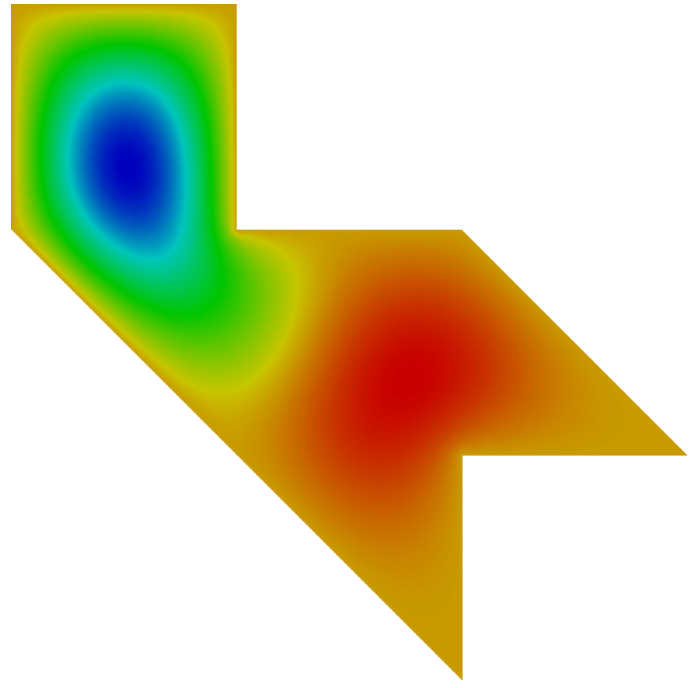


eigenmode 1

Motivation: Eigenvalue Problems

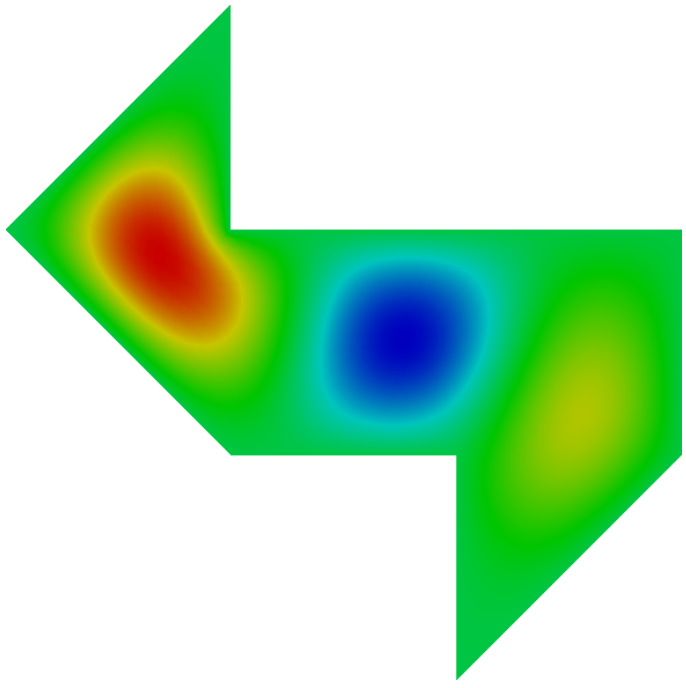


eigenmode 2

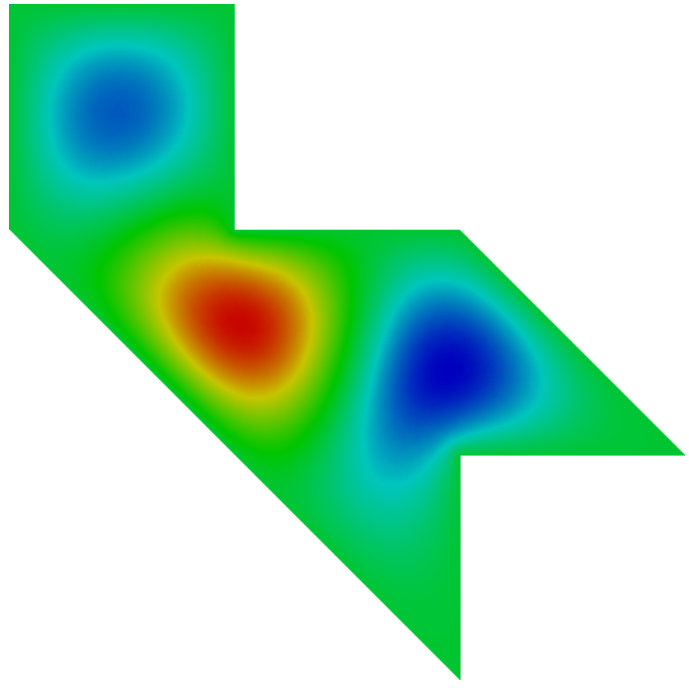


eigenmode 2

Motivation: Eigenvalue Problems

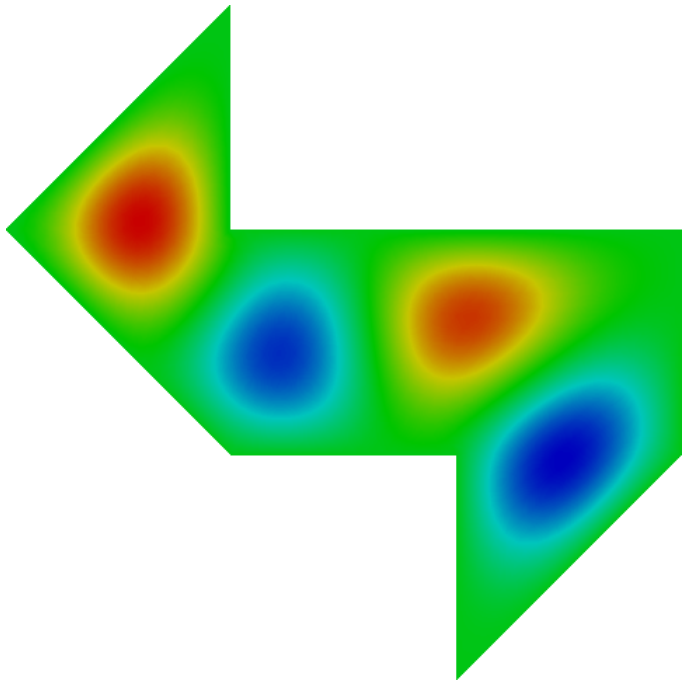


eigenmode 3

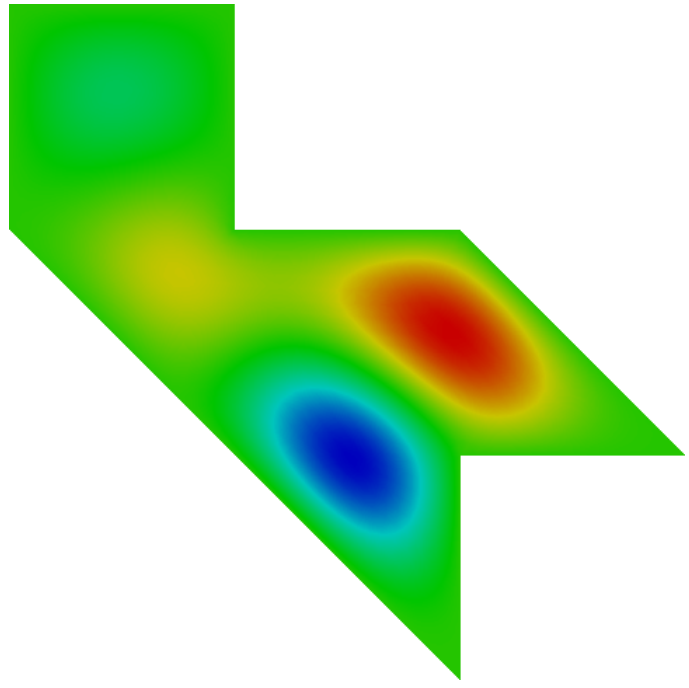


eigenmode 3

Motivation: Eigenvalue Problems

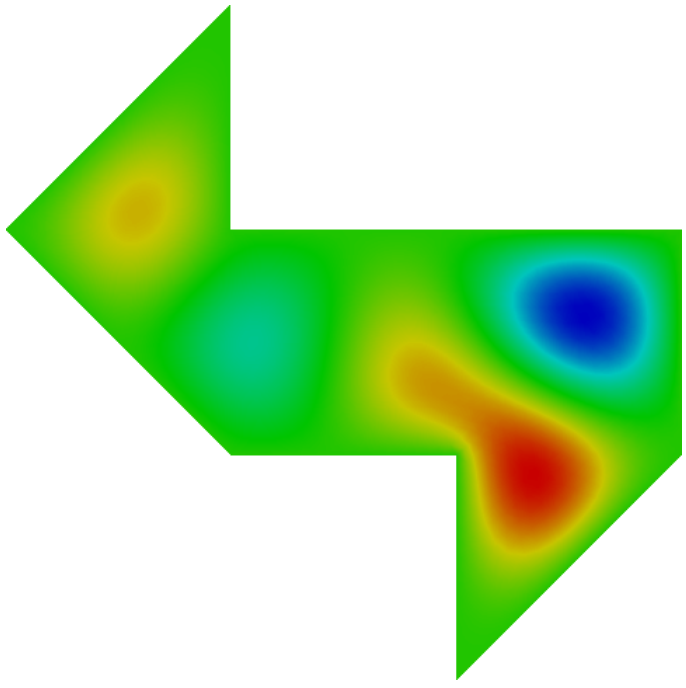


eigenmode 4

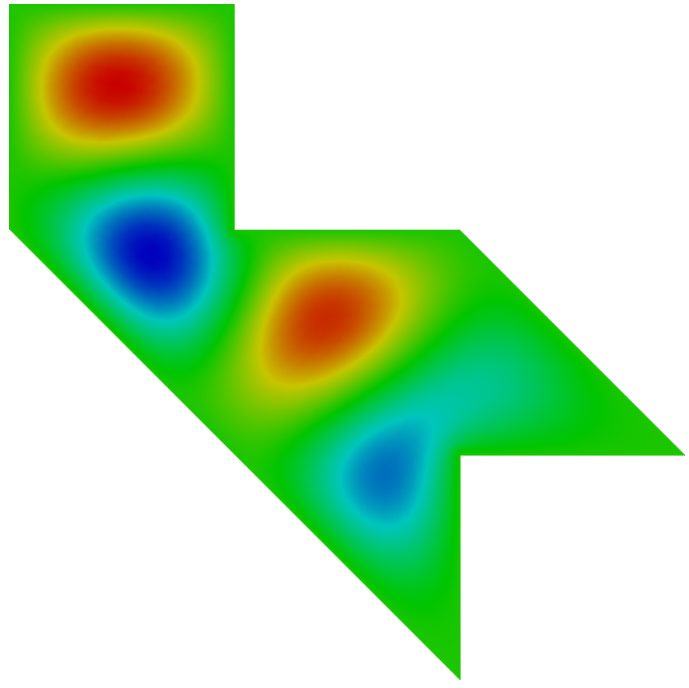


eigenmode 4

Motivation: Eigenvalue Problems



eigenmode 5



eigenmode 5

Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors of real-valued matrices can be complex

Hence in this Unit we will generally work with complex-valued matrices and vectors, $A \in \mathbb{C}^{n \times n}$, $v \in \mathbb{C}^n$

For $A \in \mathbb{C}^{n \times n}$, we shall consider the eigenvalue problem: find $(\lambda, v) \in \mathbb{C} \times \mathbb{C}^n$ such that

$$\begin{aligned} Av &= \lambda v, \\ \|v\|_2 &= 1 \end{aligned}$$

Note that for $v \in \mathbb{C}^n$, $\|v\|_2 \equiv \left(\sum_{k=1}^n |v_k|^2\right)^{1/2}$, where $|\cdot|$ is the modulus of a complex number

Eigenvalues and Eigenvectors

This problem can be reformulated as:

$$(A - \lambda \mathrm{I})v = 0$$

We know this system has a solution if and only if $(A - \lambda \mathrm{I})$ is singular, hence we must have

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

$p(z) \equiv \det(A - z\mathrm{I})$ is a degree n polynomial, called the characteristic polynomial of A

The eigenvalues of A are exactly the roots of the characteristic polynomial

Characteristic Polynomial

By the fundamental theorem of algebra, we can factorize $p(z)$ as

$$p(z) = c_n(z - \lambda_1)(z - \lambda_2)\cdots(z - \lambda_n),$$

where the roots $\lambda_i \in \mathbb{C}$ need not be distinct

Note also that complex eigenvalues of a matrix $A \in \mathbb{R}^{n\times n}$ must occur as complex conjugate pairs

That is, if $\lambda = \alpha + i\beta$ is an eigenvalue, then so is its complex conjugate $\overline{\lambda} = \alpha - i\beta$

Characteristic Polynomial

This follows from the fact that for a polynomial p with real coefficients, $p(\overline{z}) = \overline{p(z)}$ for any $z \in \mathbb{C}$:

$$p(\overline{z}) = \sum_{k=0}^{n} c_k (\overline{z})^k = \sum_{k=0}^{n} c_k \overline{z^k} = \overline{\sum_{k=0}^{n} c_k z^k} = \overline{p(z)}$$

Hence if $w \in \mathbb{C}$ is a root of p, then so is $\overline{w}$, since

$$0 = p(w) = \overline{p(w)} = p(\overline{w})$$

Companion Matrix

We have seen that every matrix has an associated characteristic polynomial

Similarly, every polynomial has an associated companion matrix

The companion matrix, C_n, of $p \in \mathbb{P}_n$ is a matrix which has eigenvalues that match the roots of p

Divide p by its leading coefficient to get a monic polynomial, i.e. with leading coefficient equal to 1 (this doesn't change the roots)

$$p_{\text{monic}}(z) = c_0 + c_1 z + \cdots + c_{n-1} z^{n-1} + z^n$$

Companion Matrix

Then p_{monic} is the characteristic polynomial of the $n \times n$ companion matrix

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

Companion Matrix

We show this for the $n = 3$ case: Consider

$$p_{3,\text{monic}}(z) \equiv c_0 + c_1 z + c_2 z^2 + z^3,$$

which has companion matrix

$$C_3 \equiv \begin{bmatrix} 0 & 0 & -c_0 \\ 1 & 0 & -c_1 \\ 0 & 1 & -c_2 \end{bmatrix}$$

Recall that for a 3×3 matrix, we have

$$\det\left(\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\right) = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32}$$
$$- a_{13}a_{22}a_{31} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33}$$

Companion Matrix

Substituting entries of C_3 then gives

$$\det(z\mathrm{I} - C_3) = c_0 + c_1 z + c_2 z^2 + z^3 = p_{3,\text{monic}}(z)$$

This link between matrices and polynomials is used by Python's `roots` function

`roots` computes all roots of a polynomial by using algorithms considered in this Unit to find eigenvalues of the companion matrix

Eigenvalue Decomposition

Let λ be an eigenvalue of $A \in \mathbb{C}^{n \times n}$; the set of all eigenvalues is called the spectrum of A

The algebraic multiplicity of λ is the multiplicity of the corresponding root of the characteristic polynomial

The geometric multiplicity of λ is the number of linearly independent eigenvectors corresponding to λ

For example, for $A = \mathrm{I}$, $\lambda = 1$ is an eigenvalue with algebraic and geometric multiplicity of n

(Char. poly. for $A = \mathrm{I}$ is $p(z) = (z-1)^n$, and $e_i \in \mathbb{C}^n$, $i = 1, 2, \ldots, n$ are eigenvectors)

Eigenvalue Decomposition

Theorem: The geometric multiplicity of an eigenvalue is less than or equal to its algebraic multiplicity

If λ has geometric multiplicity $<$ algebraic multiplicity, then λ is said to be defective

We say a matrix is defective if it has at least one defective eigenvalue

Eigenvalue Decomposition

For example,

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

has one eigenvalue with algebraic multiplicity of 3, geometric multiplicity of 1

```
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> a=np.array([[2,1,0],[0,2,1],[0,0,2]])
>>> v,d=np.linalg.eig(a)
>>> v
array([ 2.,  2.,  2.])
>>> d
array([[  1.00000000e+00,  -1.00000000e+00,   1.00000000e+00],
       [  0.00000000e+00,   4.44089210e-16,  -4.44089210e-16],
       [  0.00000000e+00,   0.00000000e+00,   1.97215226e-31]])
```

Eigenvalue Decomposition

Let $A \in \mathbb{C}^{n \times n}$ be a nondefective matrix, then it has a full set of n linearly independent eigenvectors $v_1, v_2, \ldots, v_n \in \mathbb{C}^n$

Suppose $V \in \mathbb{C}^{n \times n}$ contains the eigenvectors of A as columns, and let $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$

Then $Av_i = \lambda_i v_i$, $i = 1, 2, \ldots, n$ is equivalent to $AV = VD$

Since we assumed A is nondefective, we can invert V to obtain

$$A = VDV^{-1}$$

This is the eigendecomposition of A

This shows that for a non-defective matrix, A is diagonalized by V

Eigenvalue Decomposition

We introduce the conjugate transpose, $A^* \in \mathbb{C}^{n \times m}$, of a matrix $A \in \mathbb{C}^{m \times n}$

$$(A^*)_{ij} = \overline{A_{ji}}, \quad i = 1, 2, \ldots, m, \; j = 1, 2, \ldots, n$$

A matrix is said to be hermitian if $A = A^*$ (this generalizes matrix symmetry)

A matrix is said to be unitary if $AA^* = \mathrm{I}$ (this generalizes the concept of an orthogonal matrix)

Also, for $v \in \mathbb{C}^n$, $\|v\|_2 = \sqrt{v^* v}$

Eigenvalue Decomposition

In Python the `.T` operator performs the transpose, while the `.getH` operator performs the conjugate transpose

```
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> a=np.matrix([[1+1j,2+3j],[0,4]])
>>> a.T
matrix([[ 1.+1.j,  0.+0.j],
        [ 2.+3.j,  4.+0.j]])
>>> a.getH()
matrix([[ 1.-1.j,  0.-0.j],
        [ 2.-3.j,  4.-0.j]])
```

Eigenvalue Decomposition

In some cases, the eigenvectors of A can be chosen such that they are orthonormal

$$v_i^* v_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

In such a case, the matrix of eigenvectors, Q, is unitary, and hence A can be unitarily diagonalized

$$A = QDQ^*$$

Eigenvalue Decomposition

Theorem: A hermitian matrix is unitarily diagonalizable, and its eigenvalues are real

But hermitian matrices are not the only matrices that can be unitarily diagonalized... $A \in \mathbb{C}^{n \times n}$ is normal if

$$A^* A = A A^*$$

Theorem: A matrix is unitarily diagonalizable if and only if it is normal

Gershgorin's Theorem

Due to the link between eigenvalues and polynomial roots, in general one has to use iterative methods to compute eigenvalues

However, it is possible to gain some information about eigenvalue locations more easily from Gershgorin's Theorem

Let $D(c, r) \equiv \{x \in \mathbb{C} : |x - c| \leq r\}$ denote a disk in the complex plane centered at c with radius r

For a matrix $A \in \mathbb{C}^{n \times n}$, $D(a_{ii}, R_i)$ is called a Gershgorin disk, where

$$R_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|,$$

Gershgorin's Theorem

Theorem: All eigenvalues of $A \in \mathbb{C}^{n \times n}$ are contained within the union of the n Gershgorin disks of A

Proof: See lecture

Gershgorin's Theorem

Note that a matrix is diagonally dominant if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \quad \text{for } i = 1, 2, \ldots, n$$

It follows from Gershgorin's Theorem that a diagonally dominant matrix cannot have a zero eigenvalue, hence must be invertible

For example, the finite difference discretization matrix of the differential operator $-\Delta + \mathrm{I}$ is diagonally dominant

In 2-dimensions, $(-\Delta + \mathrm{I})u = -u_{xx} - u_{yy} + u$

Each row of the corresponding discretization matrix contains diagonal entry $4/h^2 + 1$, and four off-diagonal entries of $-1/h^2$

Sensitivity of Eigenvalue Problems

We shall now consider the sensitivity of the eigenvalues to perturbations in the matrix A

Suppose A is nondefective, and hence $A = VDV^{-1}$

Let δA denote a perturbation of A, and let $E \equiv V^{-1}\delta AV$, then

$$V^{-1}(A + \delta A)V = V^{-1}AV + V^{-1}\delta AV = D + E$$

Sensitivity of Eigenvalue Problems

For a nonsingular matrix X, the map $A \rightarrow X^{-1}AX$ is called a similarity transformation of A

Theorem: A similarity transformation preserves eigenvalues

Proof: We can equate the characteristic polynomials of A and $X^{-1}AX$ (denoted $p_A(z)$ and $p_{X^{-1}AX}(z)$, respectively) as follows:

$$\begin{aligned} p_{X^{-1}AX}(z) &= \det(z\mathrm{I} - X^{-1}AX) \\ &= \det(X^{-1}(z\mathrm{I} - A)X) \\ &= \det(X^{-1})\det(z\mathrm{I} - A)\det(X) \\ &= \det(z\mathrm{I} - A) \\ &= p_A(z), \end{aligned}$$

where we have used the identities $\det(AB) = \det(A)\det(B)$, and $\det(X^{-1}) = 1/\det(X)$ $\square$

Sensitivity of Eigenvalue Problems

The identity $V^{-1}(A + \delta A)V = D + E$ is a similarity transformation

Therefore $A + \delta A$ and $D + E$ have the same eigenvalues

Let λ_k, $k = 1, 2, \ldots, n$ denote the eigenvalues of A, and $\tilde{\lambda}$ denote an eigenvalue of $A + \delta A$

Then for some $w \in \mathbb{C}^n$, $(\tilde{\lambda}, w)$ is an eigenpair of $(D + E)$, *i.e.*

$$(D + E)w = \tilde{\lambda}w$$

Sensitivity of Eigenvalue Problems

This can be rewritten as

$$w = (\tilde{\lambda}\mathrm{I} - D)^{-1} E w$$

This is a promising start because:

- we want to bound $|\tilde{\lambda} - \lambda_k|$ for some k
- $(\tilde{\lambda}\mathrm{I} - D)^{-1}$ is a diagonal matrix with entries $1/(\tilde{\lambda} - \lambda_k)$ on the diagonal

Sensitivity of Eigenvalue Problems

Taking norms yields

$$\|w\|_2 \leq \|(\tilde{\lambda}\mathrm{I} - D)^{-1}\|_2 \|E\|_2 \|w\|_2,$$

or

$$\|(\tilde{\lambda}\mathrm{I} - D)^{-1}\|_2^{-1} \leq \|E\|_2$$

Note that the norm of a diagonal matrix is given by its largest entry (in abs. val.)[5]

$$\begin{aligned}
\max_{v \neq 0} \frac{\|Dv\|}{\|v\|} &= \max_{v \neq 0} \frac{\|(D_{11}v_1, D_{22}v_2, \ldots, D_{nn}v_n)\|}{\|v\|} \\
&\leq \left\{ \max_{i=1,2,\ldots,n} |D_{ii}| \right\} \max_{v \neq 0} \frac{\|v\|}{\|v\|} \\
&= \max_{i=1,2,\ldots,n} |D_{ii}|
\end{aligned}$$

[5] This holds for any induced matrix norm, not just the 2-norm

Sensitivity of Eigenvalue Problems

Hence $\|(\tilde{\lambda}\mathrm{I} - D)^{-1}\|_2 = 1/|\tilde{\lambda} - \lambda_{k^*}|$, where λ_{k^*} is the eigenvalue of A closest to $\tilde{\lambda}$

Therefore it follows from $\|(\tilde{\lambda}\mathrm{I} - D)^{-1}\|_2^{-1} \leq \|E\|_2$ that

$$\begin{aligned} |\tilde{\lambda} - \lambda_{k^*}| &= \|(\tilde{\lambda}\mathrm{I} - D)^{-1}\|_2^{-1} \\ &\leq \|E\|_2 \\ &= \|V^{-1}\delta A V\|_2 \\ &\leq \|V^{-1}\|_2 \|\delta A\|_2 \|V\|_2 \\ &= \mathrm{cond}(V)\|\delta A\|_2 \end{aligned}$$

This result is known as the Bauer–Fike Theorem

Sensitivity of Eigenvalue Problems

Hence suppose we compute the eigenvalues, $\tilde{\lambda}_i$, of the perturbed matrix $A + \delta A$

Then Bauer–Fike tells us that each $\tilde{\lambda}_i$ must reside in a disk of radius $\text{cond}(V)\|\delta A\|_2$ centered on some eigenvalue of A

If V is poorly conditioned, then even for small perturbations δA, the disks can be large: sensitivity to perturbations

If A is normal then $\text{cond}(V) = 1$, in which case the Bauer–Fike disk radius is just $\|\delta A\|_2$

Sensitivity of Eigenvalue Problems

Note that a limitation of Bauer–Fike is that it does not tell us which disk $\tilde{\lambda}_i$ will reside in

Therefore, this doesn't rule out the possibility of, say, all $\tilde{\lambda}_i$ clustering in just one Bauer–Fike disk

In the case that A and $A + \delta A$ are hermitian, we have a stronger result

Sensitivity of Eigenvalue Problems

Weyl's Theorem: Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ and $\tilde{\lambda}_1 \leq \tilde{\lambda}_2 \leq \cdots \leq \tilde{\lambda}_n$ be the eigenvalues of hermitian matrices A and $A+\delta A$, respectively. Then $\max_{i=1,\ldots,n} |\lambda_i - \tilde{\lambda}_i| \leq \|\delta A\|_2$.

Hence in the hermitian case, each perturbed eigenvalue must be in the disk[6] of its corresponding unperturbed eigenvalue!

[6] In fact, eigenvalues of a hermitian matrix are real, so disk here is actually an interval in $\mathbb{R}$

Sensitivity of Eigenvalue Problems

The Bauer–Fike Theorem relates to perturbations of the whole spectrum

We can also consider perturbations of individual eigenvalues

Suppose, for simplicity, that $A \in \mathbb{C}^{n \times n}$ is symmetric, and consider the perturbed eigenvalue problem

$$(A + E)(v + \Delta v) = (\lambda + \Delta\lambda)(v + \Delta v)$$

Expanding this equation, dropping second order terms, and using $Av = \lambda v$ gives

$$A\Delta v + Ev \approx \Delta\lambda v + \lambda\Delta v$$

Sensitivity of Eigenvalue Problems

Premultiply $A\Delta v + Ev \approx \Delta\lambda v + \lambda\Delta v$ by v^* to obtain

$$v^*A\Delta v + v^*Ev \approx \Delta\lambda v^*v + \lambda v^*\Delta v$$

Noting that

$$v^*A\Delta v = (v^*A\Delta v)^* = \Delta v^*Av = \lambda\Delta v^*v = \lambda v^*\Delta v$$

leads to

$$v^*Ev \approx \Delta\lambda v^*v, \quad \text{or} \quad \Delta\lambda = \frac{v^*Ev}{v^*v}$$

Sensitivity of Eigenvalue Problems

Finally, we obtain

$$|\Delta\lambda| \approx \frac{|v^* E v|}{\|v\|_2^2} \leq \frac{\|v\|_2 \|Ev\|_2}{\|v\|_2^2} = \|E\|_2,$$

so that $|\Delta\lambda| \lesssim \|E\|_2$

We observe that

- perturbation bound does not depend on $\mathrm{cond}(V)$ when we consider only an individual eigenvalue
- this individual eigenvalue perturbation bound is asymptotic; it is rigorous only in the limit that the perturbations $\to 0$

Algorithms for Eigenvalue Problems

Power Method

Power Method

The power method is perhaps the simplest eigenvalue algorithm

It finds the eigenvalue of $A \in \mathbb{C}^{n \times n}$ with largest modulus

1: choose $x_0 \in \mathbb{C}^n$ arbitrarily
2: **for** $k = 1, 2, \ldots$ **do**
3: $\quad x_k = Ax_{k-1}$
4: **end for**

Question: How does this algorithm work?

Power Method

Assuming A is nondefective, then the eigenvectors $v_1, v_2, \ldots, v_n$ provide a basis for $\mathbb{C}^n$

Therefore there exist coefficients α_i such that $x_0 = \sum_{j=1}^{n} \alpha_j v_j$

Then, we have

$$\begin{aligned} x_k &= Ax_{k-1} = A^2 x_{k-2} = \cdots = A^k x_0 \\ &= A^k \left(\sum_{j=1}^{n} \alpha_j v_j \right) = \sum_{j=1}^{n} \alpha_j A^k v_j \\ &= \sum_{j=1}^{n} \alpha_j \lambda_j^k v_j \\ &= \lambda_n^k \left(\alpha_n v_n + \sum_{j=1}^{n-1} \alpha_j \left[\frac{\lambda_j}{\lambda_n} \right]^k v_j \right) \end{aligned}$$

Power Method

Then if $|\lambda_n| > |\lambda_j|$, $1 \leq j < n$, we see that $x_k \to \lambda_n^k \alpha_n v_n$ as $k \to \infty$

This algorithm converges linearly: the error terms are scaled by a factor at most $|\lambda_{n-1}|/|\lambda_n|$ at each iteration

Also, we see that the method converges faster if λ_n is well-separated from the rest of the spectrum

Power Method

However, in practice the exponential factor λ_n^k could cause overflow or underflow after relatively few iterations

Therefore the standard form of the power method is actually the normalized power method

1: choose $x_0 \in \mathbb{C}^n$ arbitrarily
2: **for** $k = 1, 2, \ldots$ **do**
3: $\quad y_k = Ax_{k-1}$
4: $\quad x_k = y_k / \|y_k\|$
5: **end for**

Power Method

Convergence analysis of the normalized power method is essentially the same as the un-normalized case

Only difference is we now get an extra scaling factor, $c_k \in \mathbb{R}$, due to the normalization at each step

$$x_k = c_k \lambda_n^k \left(\alpha_n v_n + \sum_{j=1}^{n-1} \alpha_j \left[\frac{\lambda_j}{\lambda_n} \right]^k v_j \right)$$

Power Method

This algorithm directly produces the eigenvector v_n

One way to recover λ_n is to note that

$$y_k = Ax_{k-1} \approx \lambda_n x_{k-1}$$

Hence we can compare an entry of y_k and x_{k-1} to approximate λ_n

We also note two potential issues:

1. We require x_0 to have a nonzero component of v_n
2. There may be more than one eigenvalue with maximum modulus

Power Method

Issue 1:

- In practice, very unlikely that x_0 will be orthogonal to v_n
- Even if $x_0^* v_n = 0$, rounding error will introduce a component of v_n during the power iterations

Issue 2:

- We cannot ignore the possibility that there is more than one "max. eigenvalue"
- In this case x_k would converge to a member of the corresponding eigenspace

Power Method

An important idea in eigenvalue computations is to consider the "shifted" matrix $A - \sigma \mathrm{I}$, for $\sigma \in \mathbb{R}$

We see that

$$(A - \sigma \mathrm{I})v_i = (\lambda_i - \sigma)v_i$$

and hence the spectrum of $A - \sigma \mathrm{I}$ is shifted by $-\sigma$, and the eigenvectors are the same

For example, if all the eigenvalues are real, a shift can be used with the power method to converge to λ_1 instead of λ_n

Power Method

Python example: Consider power method and shifted power method for

$$A = \begin{bmatrix} 4 & 1 \\ 1 & -2 \end{bmatrix},$$

which has eigenvalues $\lambda_1 = -2.1623$, $\lambda_2 = 4.1623$

Inverse Iteration

Inverse Iteration

The eigenvalues of A^{-1} are the reciprocals of the eigenvalues of A, since

$$Av = \lambda v \iff A^{-1}v = \frac{1}{\lambda}v$$

Question: What happens if we apply the power method to A^{-1}?

Inverse Iteration

Answer: We converge to the largest (in modulus) eigenvalue of A^{-1}, which is $1/\lambda_1$ (recall that λ_1 is the smallest eigenvalue of A)

This is called inverse iteration

1: choose $x_0 \in \mathbb{C}^n$ arbitrarily
2: **for** $k = 1, 2, \ldots$ **do**
3: solve $Ay_k = x_{k-1}$ for y_k
4: $x_k = y_k / \|y_k\|$
5: **end for**

Inverse Iteration

Hence inverse iteration gives λ_1 without requiring a shift

This is helpful since it may be difficult to determine what shift is required to get λ_1 in the power method

Question: What happens if we apply inverse iteration to the shifted matrix $A - \sigma \mathrm{I}$?

Inverse Iteration

The smallest eigenvalue of $A - \sigma \mathrm{I}$ is $(\lambda_{i^*} - \sigma)$, where

$$i^* = \arg \min_{i=1,2,\ldots,n} |\lambda_i - \sigma|,$$

and hence...

Answer: We converge to $\tilde{\lambda} = 1/(\lambda_{i^*} - \sigma)$, then recover λ_{i^*} via

$$\lambda_{i^*} = \frac{1}{\tilde{\lambda}} + \sigma$$

Inverse iteration with shift allows us to find the eigenvalue closest to σ

Rayleigh Quotient Iteration

Rayleigh Quotient

For the remainder of this section (Rayleigh Quotient Iteration, QR Algorithm) we will assume that $A \in \mathbb{R}^{n \times n}$ is real and symmetric[7]

The Rayleigh quotient is defined as

$$r(x) \equiv \frac{x^T A x}{x^T x}$$

If $(\lambda, v) \in \mathbb{R} \times \mathbb{R}^n$ is an eigenpair, then

$$r(v) = \frac{v^T A v}{v^T v} = \frac{\lambda v^T v}{v^T v} = \lambda$$

[7] Much of the material generalizes to complex non-hermitian matrices, but symmetric case is simpler

Rayleigh Quotient

Theorem: Suppose $A \in \mathbb{R}^{n \times n}$ is a symmetric matrix, then for any $x \in \mathbb{R}^n$ we have

$$\lambda_1 \leq r(x) \leq \lambda_n$$

Proof: We write x as a linear combination of (orthogonal) eigenvectors $x = \sum_{j=1}^{n} \alpha_j v_j$, and the lower bound follows from

$$r(x) = \frac{x^T A x}{x^T x} = \frac{\sum_{j=1}^{n} \lambda_j \alpha_j^2}{\sum_{j=1}^{n} \alpha_j^2} \geq \lambda_1 \frac{\sum_{j=1}^{n} \alpha_j^2}{\sum_{j=1}^{n} \alpha_j^2} = \lambda_1$$

The proof of the upper bound $r(x) \leq \lambda_n$ is analogous $\square$

Rayleigh Quotient

Corollary: A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive definite if and only if all of its eigenvalues are positive

Proof: ($\Rightarrow$) Suppose A is symmetric positive definite (SPD), then for any nonzero $x \in \mathbb{R}^n$, we have $x^T Ax > 0$ and hence

$$\lambda_1 = r(v_1) = \frac{v_1^T A v_1}{v_1^T v_1} > 0$$

($\Leftarrow$) Suppose A has positive eigenvalues, then for any nonzero $x \in \mathbb{R}^n$

$$x^T Ax = r(x)(x^T x) \geq \lambda_1 \|x\|_2^2 > 0$$

$\square$

Rayleigh Quotient

But also, if x is an approximate eigenvector, then $r(x)$ gives us a good approximation to the eigenvalue

This is because estimation of an eigenvalue from an approximate eigenvector is an $n \times 1$ linear least squares problem: $x\lambda \approx Ax$

$x \in \mathbb{R}^n$ is our "tall thin matrix" and $Ax \in \mathbb{R}^n$ is our right-hand side

Hence the normal equation for $x\lambda \approx Ax$ yields the Rayleigh quotient, *i.e.*

$$x^T x\lambda = x^T Ax$$

Rayleigh Quotient

Question: How accurate is the Rayleigh quotient approximation to an eigenvalue?

Let's consider r as a function of x, so $r : \mathbb{R}^n \to \mathbb{R}$

$$\begin{aligned}
\frac{\partial r(x)}{\partial x_j} &= \frac{\frac{\partial}{\partial x_j}(x^T A x)}{x^T x} - \frac{(x^T A x)\frac{\partial}{\partial x_j}(x^T x)}{(x^T x)^2} \\
&= \frac{2(Ax)_j}{x^T x} - \frac{(x^T A x)2x_j}{(x^T x)^2} \\
&= \frac{2}{x^T x}(Ax - r(x)x)_j
\end{aligned}$$

(Note that the second equation relies on the symmetry of A)

Rayleigh Quotient

Therefore

$$\nabla r(x) = \frac{2}{x^T x}(Ax - r(x)x)$$

For an eigenpair (λ, v) we have $r(v) = \lambda$ and hence

$$\nabla r(v) = \frac{2}{v^T v}(Av - \lambda v) = 0$$

This shows that eigenvectors of A are stationary points of r

Rayleigh Quotient

Suppose (λ, v) is an eigenpair of A, and let us consider a Taylor expansion of $r(x)$ about v:

$$\begin{aligned} r(x) &= r(v) + \nabla r(v)^T (x - v) \\ &\quad + \frac{1}{2}(x - v)^T H_r(v)(x - v) + \text{H.O.T.} \\ &= r(v) + \frac{1}{2}(x - v)^T H_r(v)(x - v) + \text{H.O.T.} \end{aligned}$$

Hence as $x \to v$ the error in a Rayleigh quotient approximation is

$$|r(x) - \lambda| = O(\|x - v\|_2^2)$$

That is, the Rayleigh quotient approx. to an eigenvalue squares the error in a corresponding eigenvector approx.

The QR Algorithm

The QR algorithm for computing eigenvalues is one of the best known algorithms in Numerical Analysis[8]

It was developed independently in the late 1950s by John G.F. Francis (England) and Vera N. Kublanovskaya (USSR)

The QR algorithm efficiently provides approximations for all eigenvalues/eigenvectors of a matrix

We will consider what happens when we apply the power method to a set of vectors — this will then motivate the QR algorithm

[8] Recall that here we focus on the case in which $A \in \mathbb{R}^{n \times n}$ is symmetric

The QR Algorithm

Let $x_1^{(0)}, \ldots, x_p^{(0)}$ denote p linearly independent starting vectors, and suppose we store these vectors in the columns of X_0

We can apply the power method to these vectors to obtain the following algorithm:

1: choose an $n \times p$ matrix X_0 arbitrarily
2: **for** $k = 1, 2, \ldots$ **do**
3: $\quad X_k = AX_{k-1}$
4: **end for**

The QR Algorithm

From our analysis of the power method, we see that for each $i = 1, 2, \ldots, p$:

$$\begin{aligned} x_i^{(k)} &= \left(\lambda_n^k \alpha_{i,n} v_n + \lambda_{n-1}^k \alpha_{i,n-1} v_{n-1} + \cdots + \lambda_1^k \alpha_{i,1} v_1\right) \\ &= \lambda_{n-p}^k \left(\sum_{j=n-p+1}^{n} \left(\frac{\lambda_j}{\lambda_{n-p}}\right)^k \alpha_{i,j} v_j + \sum_{j=1}^{n-p} \left(\frac{\lambda_j}{\lambda_{n-p}}\right)^k \alpha_{i,j} v_j \right) \end{aligned}$$

Then, if $|\lambda_{n-p+1}| > |\lambda_{n-p}|$, the sum in green will decay compared to the sum in blue as $k \to \infty$

Hence the columns of X_k will converge to a basis for $\text{span}\{v_{n-p+1}, \ldots, v_n\}$

The QR Algorithm

However, this method doesn't provide a good basis: each column of X_k will be very close to v_n

Therefore the columns of X_k become very close to being linearly dependent

We can resolve this issue by enforcing linear independence at each step

The QR Algorithm

We orthonormalize the vectors after each iteration via a (reduced) QR factorization, to obtain the simultaneous iteration:

1: choose $n \times p$ matrix Q_0 with orthonormal columns
2: **for** $k = 1, 2, \ldots$ **do**
3: $\quad X_k = A\hat{Q}_{k-1}$
4: $\quad \hat{Q}_k \hat{R}_k = X_k$
5: **end for**

The column spaces of $\hat{Q}_k$ and X_k in line 4 are the same

Hence columns of $\hat{Q}_k$ converge to orthonormal basis for $\text{span}\{v_{n-p+1}, \ldots, v_n\}$

The QR Algorithm

In fact, we don't just get a basis for span$\{v_{n-p+1}, \ldots, v_n\}$, we get the eigenvectors themselves!

Theorem: The columns of $\hat{Q}_k$ converge to the p dominant eigenvectors of A

We will not discuss the full proof, but we note that this result is not surprising since:

- the eigenvectors of a symmetric matrix are orthogonal
- columns of $\hat{Q}_k$ converge to an orthogonal basis for span$\{v_{n-p+1}, \ldots, v_n\}$

Simultaneous iteration approximates eigenvectors, we obtain eigenvalues from the Rayleigh quotient $\hat{Q}^T A \hat{Q} \approx \text{diag}(\lambda_1, \ldots, \lambda_n)$

The QR Algorithm

With $p = n$, the simultaneous iteration will approximate all eigenpairs of A

We now show a more convenient reorganization of the simultaneous iteration algorithm

We shall require some extra notation: the Q and R matrices arising in the simultaneous iteration will be underlined $\underline{Q}_k$, $\underline{R}_k$

(As we will see shortly, this is to distinguish between the matrices arising in the two different formulations...)

The QR Algorithm

Define[9] the k^{th} Rayleigh quotient matrix: $A_k \equiv \underline{Q}_k^T A \underline{Q}_k$, and the QR factors Q_k, R_k as: $Q_k R_k = A_{k-1}$

Our goal is to show that $A_k = R_k Q_k$, $k = 1, 2, \ldots$

Initialize $\underline{Q}_0 = \mathrm{I} \in \mathbb{R}^{n \times n}$, then in the first simultaneous iteration we obtain $X_1 = A$ and $\underline{Q}_1 \underline{R}_1 = A$

It follows that $A_1 = \underline{Q}_1^T A \underline{Q}_1 = \underline{Q}_1^T (\underline{Q}_1 \underline{R}_1) \underline{Q}_1 = \underline{R}_1 \underline{Q}_1$

Also $Q_1 R_1 = A_0 = \underline{Q}_0^T A \underline{Q}_0 = A$, so that $Q_1 = \underline{Q}_1$, $R_1 = \underline{R}_1$, and $A_1 = R_1 Q_1$

[9] We now we use the full, rather than the reduced, QR factorization hence we omit $\hat{}$ notation

The QR Algorithm

In the second simultaneous iteration, we have $X_2 = A\underline{Q}_1$, and we compute the QR factorization $\underline{Q}_2\underline{R}_2 = X_2$

Also, using our QR factorization of A_1 gives

$$X_2 = A\underline{Q}_1 = (\underline{Q}_1\underline{Q}_1^T)A\underline{Q}_1 = \underline{Q}_1 A_1 = \underline{Q}_1(Q_2R_2),$$

which implies that $\underline{Q}_2 = \underline{Q}_1 Q_2 = Q_1 Q_2$ and $\underline{R}_2 = R_2$

Hence

$$A_2 = \underline{Q}_2^T A\underline{Q}_2 = Q_2^T\underline{Q}_1^T A\underline{Q}_1 Q_2 = Q_2^T A_1 Q_2 = Q_2^T Q_2 R_2 Q_2 = R_2 Q_2$$

The QR Algorithm

The same pattern continues for $k = 3, 4, \ldots$: we QR factorize A_k to get Q_k and R_k, then we compute $A_{k+1} = R_k Q_k$

The columns of the matrix $\underline{Q}_k = Q_1 Q_2 \cdots Q_k$ approximates the eigenvectors of A

The diagonal entries of the Rayleigh quotient matrix $A_k = \underline{Q}_k^T A \underline{Q}_k$ approximate the eigenvalues of A

(Also, due to eigenvector orthogonality for symmetric A, A_k converges to a diagonal matrix as $k \to \infty$)

The QR Algorithm

This discussion motivates the famous QR algorithm:

1: $A_0 = A$
2: **for** $k = 1, 2, \ldots$ **do**
3: $\quad Q_k R_k = A_{k-1}$
4: $\quad A_k = R_k Q_k$
5: **end for**

The QR Algorithm

Python demo: Compute eigenvalues and eigenvectors of[10]

$$A = \begin{bmatrix} 2.9766 & 0.3945 & 0.4198 & 1.1159 \\ 0.3945 & 2.7328 & -0.3097 & 0.1129 \\ 0.4198 & -0.3097 & 2.5675 & 0.6079 \\ 1.1159 & 0.1129 & 0.6079 & 1.7231 \end{bmatrix}$$

(This matrix has eigenvalues 1, 2, 3 and 4)

[10] Heath example 4.15

The QR Algorithm

We have presented the simplest version of the QR algorithm: the "unshifted" QR algorithm

In order to obtain an "industrial strength" algorithm, there are a number of other issues that need to be considered:

- convergence can be accelerated significantly by introducing shifts, as we did in inverse iteration and Rayleigh iteration
- it is more efficient to reduce A to tridiagonal form (via Householder reflectors) before applying QR algorithm
- reliable convergence criteria for the eigenvalues/eigenvectors are required

High-quality implementations, *e.g.* `LAPACK` or Python/MATLAB `eig`, handle all of these subtleties for us

Krylov Subspace Methods

We now give an overview of the role of Krylov[11] subspace methods in Scientific Computing

Given a matrix A and vector b, a Krylov sequence is the set of vectors

$$\{b, Ab, A^2b, A^3b, \ldots\}$$

The corresponding Krylov subspaces are the spaces spanned by successive groups of these vectors

$$\mathcal{K}_m(A, b) \equiv \text{span}\{b, Ab, A^2b, \ldots, A^{m-1}b\}$$

[11] Aleksey Krylov, 1863–1945, wrote a paper on this idea in 1931

Krylov Subspace Methods

Krylov subspaces are the basis for iterative methods for eigenvalue problems (and also for solving linear systems)

An important advantage: Krylov methods do not deal directly with A, but rather with matrix–vector products involving A

This is particularly helpful when A is large and sparse, since matrix–vector multiplications are relatively cheap

Also, the Krylov sequence is closely related to the power iteration, so it is not surprising that it is useful for solving eigenproblems

Arnoldi Iteration

Arnoldi Iteration

We define a matrix as being in Hessenberg form in the following way:

- A is called upper-Hessenberg if $a_{ij} = 0$ for all $i > j + 1$
- A is called lower-Hessenberg if $a_{ij} = 0$ for all $j > i + 1$

The Arnoldi iteration is a Krylov subspace iterative method that reduces A to upper-Hessenberg form

As we'll see, we can then use this simpler form to approximate some eigenvalues of A

Arnoldi Iteration

For $A \in \mathbb{C}^{n \times n}$, we want to compute $A = QHQ^*$, where H is upper Hessenberg and Q is unitary (*i.e.* $QQ^* = \mathrm{I}$)

However, we suppose that n is huge! Hence we do not try to compute the full factorization

Instead, let us consider just the first $m \ll n$ columns of the factorization $AQ = QH$

Therefore, on the left-hand side, we only need the matrix $Q_m \in \mathbb{C}^{n \times m}$:

$$Q_m = \left[\begin{array}{c|c|c|c} & & & \\ q_1 & q_2 & \cdots & q_m \\ & & & \end{array}\right]$$

Arnoldi Iteration

On the right-hand side, we only need the first m columns of H

More specifically, due to upper-Hessenberg structure, we only need $\widetilde{H}_m$, which is the $(m+1) \times m$ upper-left section of H:

$$\widetilde{H}_m = \begin{bmatrix} h_{11} & & \cdots & h_{1m} \\ h_{21} & h_{22} & & \\ & \ddots & \ddots & \vdots \\ & & h_{m,m-1} & h_{mm} \\ & & & h_{m+1,m} \end{bmatrix}$$

$\widetilde{H}_m$ only interacts with the first $m+1$ columns of Q, hence we have

$$AQ_m = Q_{m+1}\widetilde{H}_m$$

Arnoldi Iteration

$$\begin{bmatrix} \\ A \\ \\ \end{bmatrix} \left[\begin{array}{c|c|c} & & \\ q_1 & \cdots & q_m \\ & & \end{array}\right] = \left[\begin{array}{c|c|c} & & \\ q_1 & \cdots & q_{m+1} \\ & & \end{array}\right] \begin{bmatrix} h_{11} & \cdots & h_{1m} \\ h_{21} & \cdots & h_{2m} \\ & \ddots & \vdots \\ & & h_{m+1,m} \end{bmatrix}$$

The m^{th} column can be written as

$$Aq_m = h_{1m}q_1 + \cdots + h_{mm}q_m + h_{m+1,m}q_{m+1}$$

Or, equivalently

$$q_{m+1} = (Aq_m - h_{1m}q_1 - \cdots - h_{mm}q_m)/h_{m+1,m}$$

Arnoldi iteration is just the Gram–Schmidt method that constructs the h_{ij} and the (orthonormal) vectors q_j, $j = 1, 2, \ldots$

Arnoldi Iteration

1: choose b arbitrarily, then $q_1 = b/\|b\|_2$
2: **for** $m = 1, 2, 3, \ldots$ **do**
3: $\quad v = Aq_m$
4: $\quad$ **for** $j = 1, 2, \ldots, m$ **do**
5: $\quad\quad h_{jm} = q_j^* v$
6: $\quad\quad v = v - h_{jm} q_j$
7: $\quad$ **end for**
8: $\quad h_{m+1,m} = \|v\|_2$
9: $\quad q_{m+1} = v/h_{m+1,m}$
10: **end for**

This is akin to the modified Gram–Schmidt method because the updated vector v is used in line 5 (vs. the "raw vector" Aq_m)

Also, we only need to evaluate Aq_m and perform some vector operations in each iteration

Arnoldi Iteration

The Arnoldi iteration is useful because the q_j form orthonormal bases of the successive Krylov spaces

$$\mathcal{K}_m(A, b) = \text{span}\{b, Ab, \ldots, A^{m-1}b\} = \text{span}\{q_1, q_2, \ldots, q_m\}$$

We expect $\mathcal{K}_m(A, b)$ to provide good information about the dominant eigenvalues/eigenvectors of A

Note that this looks similar to the QR algorithm, but the QR algorithm was based on QR factorization of

$$\left[\begin{array}{c|c|c|c} & & & \\ A^k e_1 & A^k e_2 & \ldots & A^k e_n \\ & & & \end{array}\right]$$

Arnoldi Iteration

Question: How do we find eigenvalues from the Arnoldi iteration?

Let $H_m = Q_m^* A Q_m$ be the $m \times m$ matrix obtained by removing the last row from $\widetilde{H}_m$

Answer: At each step m, we compute the eigenvalues of the Hessenberg matrix H_m (via, say, the QR algorithm)[12]

This provides estimates for m eigenvalues/eigenvectors ($m \ll n$) called Ritz values, Ritz vectors, respectively

Just as with the power method, the Ritz values will typically converge to extreme eigenvalues of the spectrum

[12] This is how `eigs` in Python/Matlab works

Arnoldi Iteration

We now examine why eigenvalues of H_m approximate extreme eigenvalues of A

Let[13] $\mathbb{P}^m_{\text{monic}}$ denote the monic polynomials of degree m

Theorem: The characteristic polynomial of H_m is the unique solution of the approximation problem: find $p \in \mathbb{P}^m_{\text{monic}}$ such that

$$\|p(A)b\|_2 = \text{minimum}$$

Proof: See Trefethen & Bau

[13] Recall that a monic polynomial has coefficient of highest order term of 1

Arnoldi Iteration

This theorem implies that Ritz values (*i.e.* eigenvalues of H_m) are the roots of the optimal polynomial

$$p^* = \arg \min_{p \in \mathbb{P}^m_{\text{monic}}} \|p(A)b\|_2$$

Now, let's consider what p^* should look like in order to minimize $\|p(A)b\|_2$

We can illustrate the important ideas with a simple case, suppose:

- A has only m ($\ll n$) distinct eigenvalues
- $b = \sum_{j=1}^{m} \alpha_j v_j$, where v_j is an eigenvector corresponding to λ_j

Arnoldi Iteration

Then, for $p \in \mathbb{P}^m_{\text{monic}}$, we have

$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + x^m$$

for some coefficients $c_0, c_1, \ldots, c_{m-1}$

Applying this polynomial to a matrix A gives

$$\begin{aligned}
p(A)b &= \left(c_0 \mathrm{I} + c_1 A + c_2 A^2 + \cdots + A^m\right) b \\
&= \sum_{j=1}^{m} \alpha_j \left(c_0 \mathrm{I} + c_1 A + c_2 A^2 + \cdots + A^m\right) v_j \\
&= \sum_{j=1}^{m} \alpha_j \left(c_0 + c_1 \lambda_j + c_2 \lambda_j^2 + \cdots + \lambda_j^m\right) v_j \\
&= \sum_{j=1}^{m} \alpha_j p(\lambda_j) v_j
\end{aligned}$$

Arnoldi Iteration

Then the polynomial $p^* \in \mathbb{P}^m_{\text{monic}}$ with roots at $\lambda_1, \lambda_2, \ldots, \lambda_m$ minimizes $\|p(A)b\|_2$, since $\|p^*(A)b\|_2 = 0$

Hence, in this simple case the Arnoldi method finds p^* after m iterations

The Ritz values after m iterations are then exactly the m distinct eigenvalues of A

Arnoldi Iteration

Suppose now that there are more than m distinct eigenvalues (as is generally the case in practice)

It is intuitive that in order to minimize $\|p(A)b\|_2$, p^* should have roots close to the dominant eigenvalues of A

Also, we expect Ritz values to converge more rapidly for extreme eigenvalues that are well-separated from the rest of the spectrum

(We'll see a concrete example of this for a symmetric matrix A shortly)

Lanczos Iteration

Lanczos Iteration

Lanczos iteration is the Arnoldi iteration in the special case that A is hermitian

However, we obtain some significant computational savings in this special case

Let us suppose for simplicity that A is symmetric with real entries, and hence has real eigenvalues

Then $H_m = Q_m^T A Q_m$ is also symmetric $\implies$ Ritz values (*i.e.* eigenvalue estimates) are also real

Lanczos Iteration

Also, we can show that H_m is tridiagonal: Consider the ij entry of H_m, $h_{ij} = q_i^T A q_j$

Recall first that $\{q_1, q_2, \ldots, q_j\}$ is an orthonormal basis for $\mathcal{K}_j(A, b)$

Then we have $Aq_j \in \mathcal{K}_{j+1}(A, b) = \text{span}\{q_1, q_2, \ldots, q_{j+1}\}$, and hence $h_{ij} = q_i^T(Aq_j) = 0$ for $i > j + 1$ since

$$q_i \perp \text{span}\{q_1, q_2, \ldots, q_{j+1}\}, \text{ for } i > j + 1$$

Also, since H_m is symmetric, we have $h_{ij} = h_{ji} = q_j^T(Aq_i)$, which implies $h_{ij} = 0$ for $j > i + 1$, by the same reasoning as above

Lanczos Iteration

Since H_m is now tridiagonal, we shall write it as

$$T_m = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{m-1} \\ & & & \beta_{m-1} & \alpha_m \end{bmatrix}$$

The consequence of tridiagonality: Lanczos iteration is much cheaper than Arnoldi iteration!

Lanczos Iteration

The inner loop in Lanczos iteration only runs from $m-1$ to m, instead of 1 to m as in Arnoldi

This is due to the three-term recurrence at step m:

$$Aq_m = \beta_{m-1} q_{m-1} + \alpha_m q_m + \beta_m q_{m+1}$$

(This follows from our discussion of the Arnoldi case, with $\widetilde{T}_m$ replacing $\widetilde{H}_m$)

As before, we rearrange this to give

$$q_{m+1} = (Aq_m - \beta_{m-1} q_{m-1} - \alpha_m q_m)/\beta_m$$

Lanczos Iteration

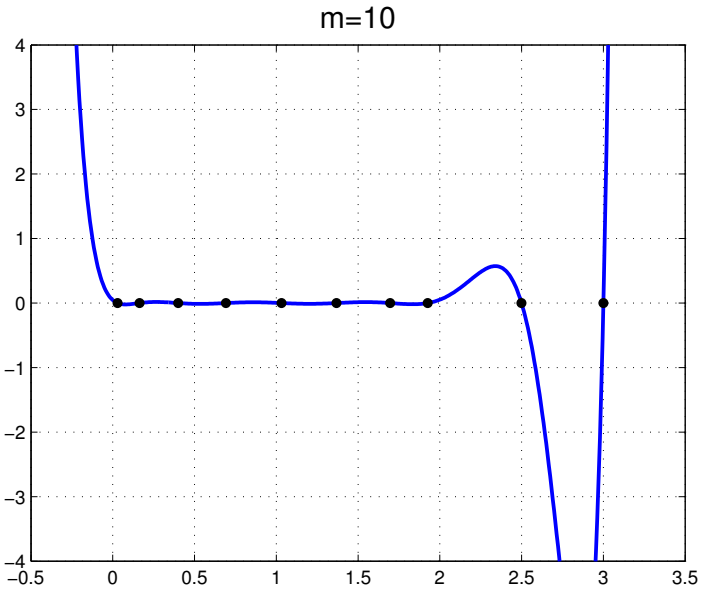
Which leads to the Lanczos iteration

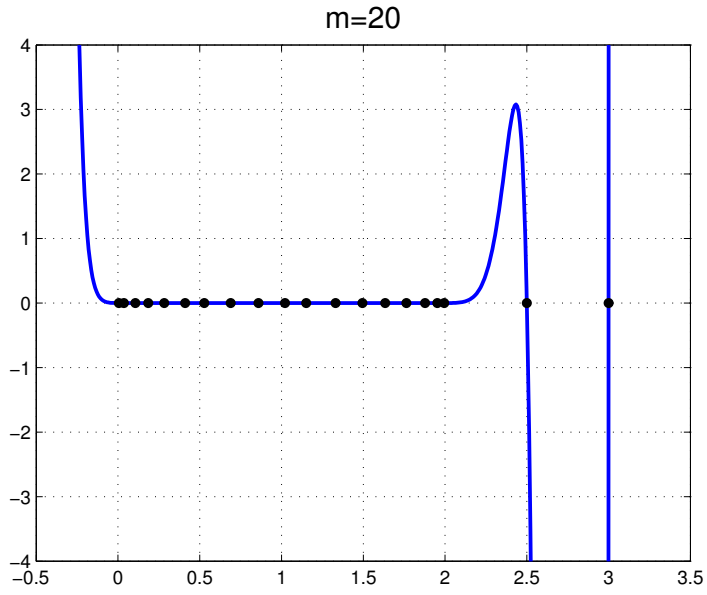
1: $\beta_0 = 0$, $q_0 = 0$
2: choose b arbitrarily, then $q_1 = b/\|b\|_2$
3: **for** $m = 1, 2, 3, \ldots$ **do**
4: $\quad v = Aq_m$
5: $\quad \alpha_m = q_m^T v$
6: $\quad v = v - \beta_{m-1} q_{m-1} - \alpha_m q_m$
7: $\quad \beta_m = \|v\|_2$
8: $\quad q_{m+1} = v/\beta_m$
9: **end for**

Lanczos Iteration

Python demo: Lanczos iteration for a diagonal matrix

Lanczos Iteration





We can see that Lanczos minimizes $\|p(A)b\|_2$:

- p is uniformly small in the region of clustered eigenvalues
- roots of p match isolated eigenvalues very closely

Note that in general p will be very steep near isolated eigenvalues, hence convergence for isolated eigenvalues is rapid!

Conjugate Gradient Method

Conjugate Gradient Method

We now turn to another Krylov subspace method: the conjugate gradient method,[14] or CG

(This is a detour in this Unit since CG is not an eigenvalue algorithm)

CG is an iterative method for solving $Ax = b$ in the case that A is symmetric positive definite (SPD)

CG is the original (and perhaps most famous) Krylov subspace method, and is a mainstay of Scientific Computing

[14] Due to Hestenes and Stiefel in 1952

Conjugate Gradient Method

Recall that we discussed CG in Unit 4 — the approach in Unit 4 is also often called nonlinear conjugate gradients

Nonlinear conjugate gradients applies the approach of Hestenes and Stiefel to nonlinear unconstrained optimization

Conjugate Gradient Method

Iterative solvers (*e.g.* CG) and direct solvers (*e.g.* Gaussian elimination) for solving $Ax = b$ are fundamentally different:

- Direct solvers: In exact arithmetic, gives exact answer after finitely many steps
- Iterative solvers: In principle require infinitely many iterations, but should give accurate approximation after few iterations

Conjugate Gradient Method

Direct methods are very successful, but iterative methods are typically more efficient for very large, sparse systems

Krylov subspace methods only require matvecs and vector-vector (*e.g.* dot product) operations

Hence Krylov methods require $O(n)$ operations per iteration for sparse A, no issues with "fill-in," *etc.*

Also, iterative methods are generally better suited to parallelization, hence an important topic in supercomputing

Conjugate Gradient Method

The CG algorithm is given by

1: $x_0 = 0,\ r_0 = b,\ p_0 = r_0$
2: **for** $k = 1, 2, 3, \ldots$ **do**
3: $\quad \alpha_k = (r_{k-1}^T r_{k-1})/(p_{k-1}^T A p_{k-1})$
4: $\quad x_k = x_{k-1} + \alpha_k p_{k-1}$
5: $\quad r_k = r_{k-1} - \alpha_k A p_{k-1}$
6: $\quad \beta_k = (r_k^T r_k)/(r_{k-1}^T r_{k-1})$
7: $\quad p_k = r_k + \beta_k p_{k-1}$
8: **end for**

Conjugate Gradient Method

We shall now discuss CG in more detail — it's certainly not obvious upfront why this is a useful algorithm!

Let $x_* = A^{-1}b$ denote the exact solution, and let $e_k \equiv x_* - x_k$ denote the error at step k

Also, let $\|\cdot\|_A$ denote the norm

$$\|x\|_A \equiv \sqrt{x^T A x}$$

Conjugate Gradient Method

Theorem: The CG iterate x_k is the unique member of $\mathcal{K}_k(A, b)$ which minimizes $\|e_k\|_A$. Also, $x_k = x_*$ for some $k \leq n$.

Proof: This result relies on a set of identities which can be derived (by induction) from the CG algorithm:

(i) $\mathcal{K}_k(A, b) = \text{span}\{x_1, x_2, \ldots, x_k\} = \text{span}\{p_0, p_1 \ldots, p_{k-1}\}$
$= \text{span}\{r_0, r_1, \ldots, r_{k-1}\}$

(ii) $r_k^T r_j = 0$ for $j < k$

(iii) $p_k^T A p_j = 0$ for $j < k$

Conjugate Gradient Method

From the first identity above, it follows that $x_k \in \mathcal{K}_k(A, b)$

We will now show that x_k is the unique minimizer in $\mathcal{K}_k(A, b)$

Let $\tilde{x} \in \mathcal{K}_k(A, b)$ be another "candidate minimizer" and let $\Delta x \equiv x_k - \tilde{x}$, then

$$\begin{aligned} \|x_* - \tilde{x}\|_A^2 &= \|(x_* - x_k) + (x_k - \tilde{x})\|_A^2 \\ &= \|e_k + \Delta x\|_A^2 \\ &= (e_k + \Delta x)^T A(e_k + \Delta x) \\ &= e_k^T A e_k + 2e_k^T A\Delta x + \Delta x^T A\Delta x \end{aligned}$$

Conjugate Gradient Method

Next, let $r(x_k) = b - Ax_k$ denote the residual at step k, so that

$$r(x_k) = b - Ax_k = b - A(x_{k-1} + \alpha_k p_{k-1}) = r(x_{k-1}) - \alpha_k A p_{k-1}$$

Since $r(x_0) = b = r_0$, by induction we see that for r_k computed in line 5 of CG,

$$r_k = r_{k-1} - \alpha_k A p_{k-1}$$

we have $r_k = r(x_k),\ k = 1, 2, \ldots$

Conjugate Gradient Method

Now, recall our expression for $\|x_* - \tilde{x}\|_A^2$:

$$\|x_* - \tilde{x}\|_A^2 = e_k^T A e_k + 2e_k^T A \Delta x + \Delta x^T A \Delta x$$

and note that

$$2e_k^T A \Delta x = 2\Delta x^T A(x_* - x_k) = 2\Delta x^T (b - Ax_k) = 2\Delta x^T r_k$$

Now,

- $\Delta x = x_k - \tilde{x} \in \mathcal{K}_k(A, b)$
- from (i), we have that $\mathcal{K}_k(A, b) = \text{span}\{r_0, r_1, \ldots, r_{k-1}\}$
- from (ii), we have that $r_k \perp \text{span}\{r_0, r_1, \ldots, r_{k-1}\}$

Therefore, we have $2e_k^T A \Delta x = 2\Delta x^T r_k = 0$

Conjugate Gradient Method

This implies that,

$$\|x_* - \tilde{x}\|_A^2 = e_k^T A e_k + \Delta x^T A \Delta x \geq \|e_k\|_A^2,$$

with equality only when $\Delta x = 0$, hence $x_k \in \mathcal{K}_k(A, b)$ is the unique minimizer!

This also tells us that if $x_* \in \mathcal{K}_k(A, b)$, then $x_k = x_*$

Therefore[15] CG will converge to x_* in at most n iterations since $\mathcal{K}_k(A, b)$ is a subspace of $\mathbb{R}^n$ of dimension k $\square$

[15] Assuming exact arithmetic!

Conjugate Gradient Method

Note that the theoretical guarantee that CG will converge in n steps is of no practical use

In floating point arithmetic we will not get exact convergence to x_*

More importantly, we assume n is huge, so we want to terminate CG well before n iterations anyway

Nevertheless, the guarantee of convergence in at most n steps is of historical interest

Hestenes and Stiefel originally viewed CG as a direct method that will converge after a finite number of steps

Conjugate Gradient Method

Steps of CG are chosen to give the orthogonality properties (ii), (iii), which lead to the remarkable CG optimality property:

> CG minimizes the error over the Krylov subspace $\mathcal{K}_k(A, b)$ at step k

Question: Where did the steps in the CG algorithm come from?

Answer: It turns out that CG can be derived by developing an optimization algorithm for $\phi : \mathbb{R}^n \to \mathbb{R}$ given by

$$\phi(x) \equiv \frac{1}{2} x^T A x - x^T b$$

e.g. lines 3 and 4 in CG perform line search, line 7 gives a search direction p_k

Conjugate Gradient Method

[Aside: Note that $-\nabla\phi(x) = b - Ax = r(x)$

The name "Conjugate Gradient" then comes from the property

$$\text{(ii)} \quad \nabla\phi(x_k)^T \nabla\phi(x_j) = r_k^T r_j = 0 \text{ for } j < k$$

That is, the gradient directions are orthogonal, or "conjugate"]

Question: Why is the quadratic objective function ϕ relevant to solving $Ax = b$?

Conjugate Gradient Method

Answer: Minimizing ϕ is equivalent to minimizing $\|e_k\|_A^2$, since

$$\begin{aligned} \|e_k\|_A^2 &= (x_* - x_k)^T A(x_* - x_k) \\ &= x_k^T A x_k - 2x_k^T A x_* + x_*^T A x_* \\ &= x_k^T A x_k - 2x_k^T b + x_*^T b \\ &= 2\phi(x_k) + \text{const.} \end{aligned}$$

Hence, our argument from above shows that, at iteration k, CG solves the optimization problem

$$\min_{x \in \mathcal{K}_k(A,b)} \phi(x)$$

Conjugate Gradient Method

An important topic (that we will not cover in detail) is convergence analysis of CG: How fast does $\|e_k\|_A$ converge?

A famous result for CG is that if A has 2-norm condition number κ, then

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k$$

Hence smaller condition number implies faster convergence!

Conjugate Gradient Method

Suppose we want to terminate CG when

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq \epsilon$$

for some $\epsilon > 0$, how many CG iterations will this require?

We have the identities

$$\begin{aligned} 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k &= 2\left(\frac{\sqrt{\kappa}+(1-1)-1}{\sqrt{\kappa}+1}\right)^k \\ &= 2\left(1-\frac{2}{\sqrt{\kappa}+1}\right)^k \\ &= 2\left(1-\frac{2/\sqrt{\kappa}}{1+1/\sqrt{\kappa}}\right)^k \end{aligned}$$

Conjugate Gradient Method

And for large κ it follows that

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \approx 2\left(1-\frac{2}{\sqrt{\kappa}}\right)^k$$

Hence we terminate CG when

$$\left(1-\frac{2}{\sqrt{\kappa}}\right)^k \approx \frac{\epsilon}{2}$$

Conjugate Gradient Method

Taking logs gives

$$k \approx \log(\epsilon/2)/\log\left(1 - \frac{2}{\sqrt{\kappa}}\right) \approx \frac{1}{2}|\log(\epsilon/2)|\sqrt{\kappa}$$

where the last expression follows from the Taylor expansion:

$$\log\left(1 - \frac{2}{\sqrt{\kappa}}\right) \approx \log(1) - \frac{2}{\sqrt{\kappa}} = -\frac{2}{\sqrt{\kappa}}$$

This analysis shows that the number of CG iterations for a given tolerance ϵ grows approximately as $\sqrt{\kappa}$

Conjugate Gradient Method

For the discrete Laplacian, we have $\kappa = O(h^{-2})$,[16] hence number of CG iterations should grow as $O(\sqrt{\kappa}) = O(h^{-1})$

For $\epsilon = 10^{-4}$, we obtain the following convergence results

h	κ	CG iterations
4×10^{-2}	3.67×10^{2}	32
2×10^{-2}	1.47×10^{3}	65
1×10^{-2}	5.89×10^{3}	133
5×10^{-3}	2.36×10^{4}	272

[16] This is a standard result in finite element analysis

Conjugate Gradient Method

These results indicate that CG gets more expensive for Poisson equation as h is reduced for two reasons:

- The matrix and vectors get larger, hence each CG iteration is more expensive
- We require more iterations since the condition number gets worse

Conjugate Gradient Method: Preconditioning

The final crucial idea that we will mention is preconditioning

The idea is that we premultiply $Ax = b$ by the preconditioning matrix M to obtain the system $MAx = Mb$

The CG convergence rate will then depend on the properties of MA rather than A

We know (from Abel and Galois) that we can't transform a matrix to triangular form via finite sequence of similarity transformations

However, it is possible to "similarity transform" *e.g.* one intuitive idea is to choose $M \approx A^{-1}$ so that $MA \approx \mathrm{I}$ has a smaller condition number than A

Conjugate Gradient Method: Preconditioning

Good preconditioners must be cheap to compute, and should significantly accelerate convergence of an iterative method

Preconditioners can have a dramatic effect on convergence!

For example, preconditioning can ensure that number of CG iterations required for Poisson equation is independent of h

Preconditioning for Krylov subspace methods is a major topic in Scientific Computing: It is essential for large-scale problems!