# Numerical Methods I: Eigenvalues and eigenvectors

Georg Stadler
Courant Institute, NYU
stadler@cims.nyu.edu

November 2, 2017

Conditioning

# Eigenvalues and eigenvectors

## How hard are they to find?

For a matrix $A \in \mathbb{C}^{n \times n}$ (potentially real), we want to find $\lambda \in \mathbb{C}$ and $\boldsymbol{x} \neq 0$ such that

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

Most relevant problems:

- A symmetric (and large)
- A spd (and large)
- A stochastic matrix, i.e., all entries $0 \leq a_{ij} \leq 1$ are probabilities, and thus $\sum_j a_{ij} = 1$.

# Eigenvalues and eigenvectors

## How hard are they to find?

- This is a nonlinear problem.
- How difficult is this? Eigenvalues are the roots of the characteristic polynomial. Also, any polynomial is the characteristic polynomial of a matrix. Thus, for matrices larger than $4 \times 4$, eigenvalues cannot be computed analytically.
- Must use an iterative algorithm.

# Eigenvalues and eigenvectors

## Why useful?

- Example: Google's page rank algorithms is at its core a very big eigenvector computation with a stochastic matrix, where each webpage corresponds to a row/column, and the entries are computed from the links between web pages.
- Original page rank paper is by Google founders Page and Brin (10,000 citations, 500 billion value)
- SIAM Review paper from 2006: *The $25,000,000,000 Eigenvector: The linear Algebra behind Google.* http://dx.doi.org/10.1137/050623280

# Conditioning

Consider an algebraically simple eigenvalue $\lambda_0$ of $A$:

$$A\boldsymbol{x}_0 = \lambda_0 \boldsymbol{x}_0.$$

Then, there exists a continuously differentiable map

$$\lambda(\cdot) : A \to \lambda(A)$$

in a neighborhood of $A$ such that $\lambda(A) = \lambda_0$. The derivative is

$$\lambda'(A)C = \frac{(C\boldsymbol{x}_0, \boldsymbol{y}_0)}{(\boldsymbol{x}_0, \boldsymbol{y}_0)},$$

where $\boldsymbol{y}_0$ is an eigenvector of $A^T$ for the eigenvalue $\lambda_0$.

Why is $\lambda_0$ also an eigenvalue of $A^T$? Because char. polynomial of $A$ and $A^T$ are the same:

$$\chi_A(\lambda) = \det(\lambda I - A) = \det\left((\lambda I - A)^T\right) = \det(\lambda I - A^T) = \chi_{A^T}(\lambda)$$

# Conditioning

## Sketch of proof

Want to show: $\lambda'(A)\,C = \dfrac{(Cx_0, y_0)}{(x_0, y_0)}$

Consider perturbations of $A$: $A + tC$, $t$ small, $C \in \mathbb{K}^{n\times n}$ fixed

implicit function theorem $\Rightarrow$ $\lambda(t), x(t)$:

$(A+tC)\,x(t) = \lambda(t)\,x(t)$ for $t$ small, uses that $\lambda_0$ is single eigenvalue

for $t = 0$: $A x_0 = \lambda_0 x_0$

$\frac{\partial}{\partial t}$ of $\quad (A+tC)\,x'(t) + C x(t) = \lambda'(t)\,x(t) + \lambda(t)\,x'(t)$

$t=0$: $\quad A x'(0) + C x_0 = \lambda'(0)\,x_0 + \lambda_0 x'(0)$

inner prod with $y_0$: $\quad (Ax'(0), y_0) + (Cx_0, y_0) = \lambda'(0)(x_0, y_0) + \lambda_0 (x'(0), y_0)$

$(x'(0), A^T y_0) = (x'(0), \lambda_0 y_0)$

$\rightarrow (Cx_0, y_0) = \lambda'(0)(x_0, y_0) \Longrightarrow \lambda'(0) = \dfrac{(Cx_0, y_0)}{(x_0, y_0)} = \lambda'(A)\,C$

$\lambda(t) \triangleq A + tC \rightarrow \lambda(A+tC)$, $\lambda'(t) = \lambda'(A+tC)\,C$, $t=0$

$\operatorname{rank}\left((\lambda_i A - \lambda_j \mathrm{Id})_{ij}\right) = n$

# When is the implicit function theorem applicable?

look at

$F(x,\lambda) = Ax - \lambda x$

$\frac{\partial}{\partial \lambda} F(x,\lambda) = x$

$\frac{\partial}{\partial x} F(x,\lambda) = (A - \lambda I)$

$\Rightarrow$ Jacobian $\left[x,\ A-\lambda I\right] \in \mathbb{R}^{n \times n+1}$

If $\lambda$ is single eigenvalue, the null-space of $A-\lambda I$ is one-dimensional, and spanned by $x$. Thus, $\operatorname{rank}\left(\left[x,\ A-\lambda I\right]\right) = n$, i.e. the rank is maximal and the implicit function theorem can be applied.

If $\lambda$ has higher multiplicity, the corresponding null space of $A-\lambda I$ can have dimension $> 1$, and $\operatorname{rank}\left(\left[x,\ A-\lambda I\right]\right) < n$. Thus the implicit function theorem cannot be applied.

# Conditioning

Compute norm of $\lambda'(A)$ as linear mapping that maps

$$C \mapsto \frac{(C\boldsymbol{x}_0, \boldsymbol{y}_0)}{(\boldsymbol{x}_0, \boldsymbol{y}_0)}.$$

$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}$

Use as norm for $C$ the norm induced by the Euclidean norm:

$$\|\lambda'(A)\| = \sup_{C \neq 0} \frac{|(C\boldsymbol{x}_0, \boldsymbol{y}_0)/(\boldsymbol{x}_0, \boldsymbol{y}_0)|}{\|C\|} = \frac{\|\boldsymbol{x}_0\|\|\boldsymbol{y}_0\|}{|(\boldsymbol{x}_0, \boldsymbol{y}_0)|},$$

i.e., the inverse cosine of the angle between $\boldsymbol{x}_0, \boldsymbol{y}_0$.

$\frac{(Cx_0, y_0)}{\|C\|} \leq \|x_0\| \|y_0\|$

choose $C$ as $y_0 x_0^T \in \mathbb{R}^{n \times n}$

# Conditioning

Theorem: The absolute and relative condition numbers for computing a simple eigenvalue $\lambda_0$ are

$$\kappa_{\text{abs}} = \|\lambda'(A)\| = \frac{1}{|\cos(\sphericalangle(\boldsymbol{x}_0, \boldsymbol{y}_0))|},$$

and

$$\kappa_{\text{rel}} = \frac{\|A\|}{|\lambda_0 \cos(\sphericalangle(\boldsymbol{x}_0, \boldsymbol{y}_0))|}.$$

In particular, for normal matrices, $\kappa_{\text{abs}} = 1$. Note that finding non-simple eigenvalues is ill-posed (but can still be done).

# Power method and variants

# The power method

Choose starting point $\boldsymbol{x}^0$ and iterate

$$\boldsymbol{x}^{k+1} := A\boldsymbol{x}^k,$$

Idea: Eigenvector corresponding to largest (in absolute norm) eigenvalue will start dominating, i.e., $\boldsymbol{x}^k$ converges to eigenvector direction for largest eigenvalue $\boldsymbol{x}$. Normalize to length 1: $\boldsymbol{y}^k := \boldsymbol{x}^k / \|\boldsymbol{x}^k\|$.

- Convergence speed depends on eigenvalues
- Only finds largest eigenvalue $\lambda_{\max} = \boldsymbol{y}^T A \boldsymbol{y}$ upon convergence

Rayleigh coeff: $\frac{y^T A y}{y^T y}$ if $y$ is eigenvector, $y^T A y = y^T \lambda y = \lambda\, y^T y$

# The power method

## Convergence

Thm: Let $A \in \mathbb{R}^{n \times n}$ be symmetric and $\lambda_1$ be a simple eigenvalue with

$$|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|.$$

If $x^0$ is not orthogonal to the eigenspace of $\lambda_1$, then the power method converges to a normalized eigenvector of $A$ corresponding to $\lambda_1$.

Proof: $A$ symmetric, $\exists$ basis of eigenvectors $\eta_1, \ldots, \eta_n$,

$$x^0 = \sum_{i=1}^{n} \alpha_i \eta_i, \quad \alpha_i \in \mathbb{R}, \quad \alpha_1 \neq 0$$

$$x^k = A x^{k-1} = A^k x^0 = \sum_{i=1}^{n} \alpha_i A^k \eta_i = \sum_{i=1}^{n} \alpha_i \lambda_i^k \eta_i$$

$$= \alpha_1 \lambda_1^k \underbrace{\left( \eta_1 + \sum_{i=2}^{n} \frac{\alpha_i}{\alpha_1} \left( \frac{\lambda_i}{\lambda_1} \right)^k \eta_i \right)}_{z_k}$$

# The power method

## Convergence

$z_k$ converges to $\eta_1$ as $k \to \infty$ since $\left|\frac{\lambda_i}{\lambda_1}\right| < 1$

$$\Rightarrow \frac{x^k}{\|x^k\|} = \pm \frac{z_k}{\|z_k\|} \longrightarrow \pm \eta_1 \qquad \square.$$

Convergence speed depends on $\left|\frac{\lambda_i}{\lambda_1}\right| < 1$, if $\left|\frac{\lambda_2}{\lambda_1}\right|$

- is close to 1 $\longrightarrow$ slow convergence
- is $\ll 1$ $\longrightarrow$ fast convergence

# The power method—variants

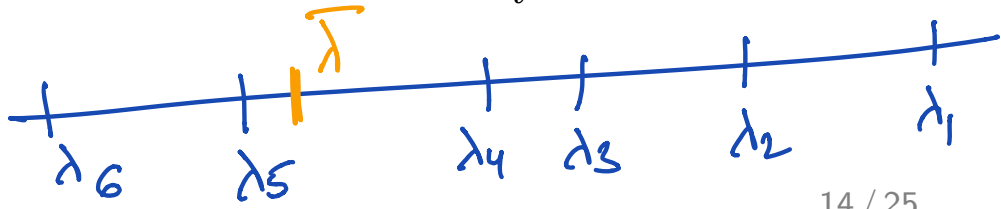
Inverse power method: Having an estimate $\bar{\lambda}$ for an eigenvalue $\lambda_i$, consider the $(A - \bar{\lambda}I)^{-1}$ which has the eigenvalues

$$(\lambda_i - \bar{\lambda})^{-1}, \quad i = 1, \ldots, n.$$

Consider the inverse power iteration

$$(A - \bar{\lambda}I)\boldsymbol{x}^{k+1} = \boldsymbol{x}^k, \quad \boldsymbol{x}^{k+1} = \boldsymbol{x}^{k+1}/\|\boldsymbol{x}^{k+1}\|$$

- Requires matrix-solve in every iteration
- Same matrix, different right hand sides (single LU or Choleski factorization)
- Convergence speed depends on how close $\bar{\lambda}$ is to $\lambda_i$.



# The power method—variants

Rayleigh quotient iteration: Accelerated version of the inverse power method using of changing shifts:

- Choose starting vector $\boldsymbol{x}^0$ with $\|\boldsymbol{x}^0\| = 1$. Compute $\lambda_0 = (\boldsymbol{x}^0)^T A \boldsymbol{x}^0$.
- For $i = 0, 1, \ldots$ do

$$(A - \bar{\lambda}_k I)\boldsymbol{x}^{k+1} = \boldsymbol{x}^k, \quad \boldsymbol{y}^{k+1} = \boldsymbol{x}^{k+1}/\|\boldsymbol{x}^{k+1}\|.$$

- Compute $\lambda_{k+1} = (\boldsymbol{y}^{k+1})^T A \boldsymbol{y}^{k+1}$, and go back.

# The QR algorithm

# The QR algorithm

The QR algorithm for finding eigenvalues is as follows ($A^0 := A$), and for $k = 0, 1, \ldots$:

- Compute QR decomposition of $A^k$, i.e., $A^k = QR$.
- $A^{k+1} := RQ$, $k := k + 1$ and go back.

Why should that converge to something useful?

# The QR algorithm

- Similarity transformations do not change the eigenvalues, i.e., if $B$ is invertible, then

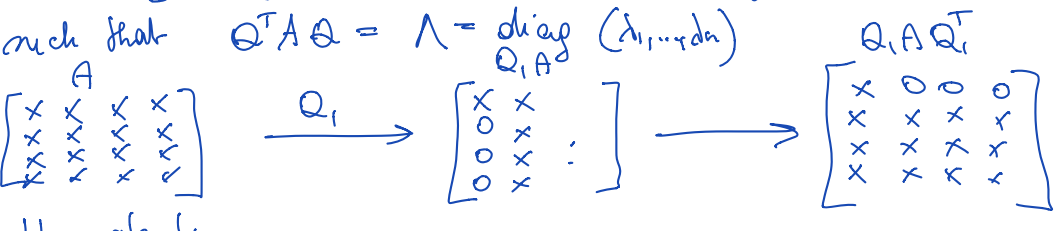
$$A \text{ and } P^{-1}AP$$

  have the same eigenvalues.
- $QA^{k+1}Q^T = QRQQ^T = QR = A^k$, i.e., the iterates $A^k$ in the QR algorithm have the same eigenvalues.
- The algorithm is closely related to the Rayleigh coefficient method.
- The algorithms is expensive (QR-decomposition is $O(n^3)$).
- Convergence can be slow.

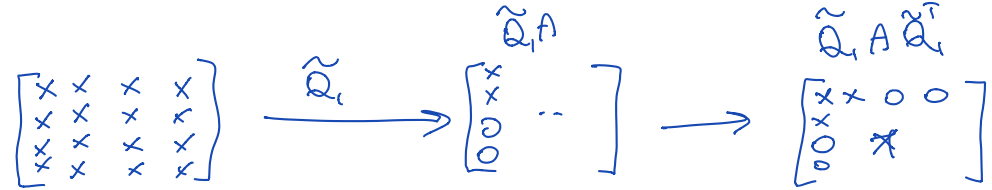
# QR algorithm and Hessenberg matrices

Find a matrix form that is invariant under the QR algorithm:

Let's try using orthogonal transformations (Givens rotations) such that $Q^T A Q = \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$

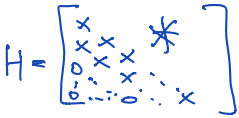



How about:



# QR algorithm and Hessenberg matrices

Idea: Find a matrix format that is preserved in the QR-algorithm. Hessenberg matrices $H$ are matrices for which $H_{i,j} = 0$ if $i > j + 1$.



- Hessenberg matrices remain Hessenberg in the QR algorithm.
- An iteration of the QR-algorithm with a Hessenberg matrix requires $O(n^2)$ flops.

Algorithm:

1. Use Givens rotations to transfer $A$ into Hessenberg form. Use transpose operations on right hand side (similarity transformation).
2. Use QR algorithm for the Hessenberg matrix.

# The QR algorithm for symmetric matrices

# QR algorithm for symmetric matrices

Let's consider symmetric matrices $A$. Then the Hessenberg form (after application of Given rotations from both sides) is tridiagonal.
Theorem: For a symmetric matrix with distinct eigenvalues

$$|\lambda_1| > \ldots |\lambda_n| > 0, \quad \Lambda = \mathsf{diag}(\lambda_1, \ldots, \lambda_n)$$

holds:

1. $\lim_{k\to\infty} Q_k = I$,
2. $\lim_{k\to\infty} R_k = \Lambda$,
3. $a_{ij}^k = O\left(\left|\frac{\lambda_i}{\lambda_j}\right|^k\right)$ for $i > j$.

# QR algorithm for symmetric matrices

- The method also converges in the presence of multiple eigenvalues. Only when $\lambda_i = -\lambda_{i+1}$, the corresponding block does not become diagonal.
- One can speed up the method by introducing a shift operator:
  - $A_k - \sigma_k I = QR$.
  - $A_{k+1} = RQ + \sigma_k I$.

Again, the shift can be updated during the iteration.

# QR algorithm for symmetric matrices

## Summary

A Symmetric : Algorithm:

1. Make A tridiagonal using Givens rotations, i.e. find P orthogonal such that

$$A_1 = PAP^T = \begin{bmatrix} \times & \times & 0 & 0 \\ \times & \times & \times & 0 \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}$$

$A_1$ has the same eigenvalues as $A$

$P$'s are products of Givens rotations $O(n^3)$

2. Apply QR algorithm to $A_1$ :

$$\Omega A_1 \Omega^T \simeq \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$$

$\Omega$: products of $Q_1 Q_2 \ldots$ arising in QR algorithm

# QR algorithm for symmetric matrices

## Summary

eigenvectors:

$$\Lambda = \Omega A_1 \Omega^T = \underbrace{\Omega P}_{Q^T} A P^T \Omega^T$$

$$= Q^T A Q$$

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

$$\Rightarrow AQ = Q\Lambda, \text{ i.e. } Aq_i = \lambda_i q_i$$

$q_i$ are the colums of $Q$

# QR algorithm for symmetric matrices

## Summary

Complexity: Convert to Hessenberg form using Givens rotations: $4/3n^3$ flops; each QR iteration: $O(n^2)$ flops. Overall, convergence is dominated by the reduction to tridiagonal form.

This method finds all eigenvalues (of a symmetric matrix).

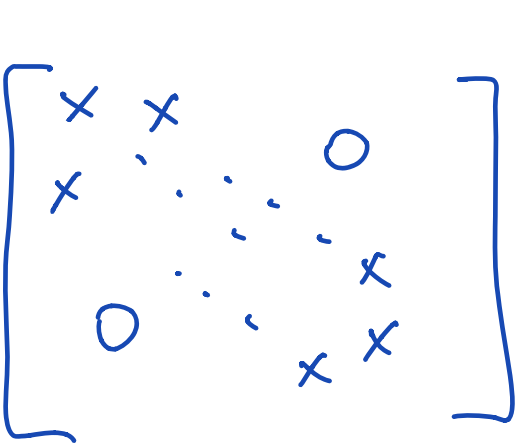
The corresponding eigenvectors can be found from the algorithm as well:

$$\Omega A_1 \Omega^T \sim \Lambda$$

with products of Givens rotations $\Omega$. If the original transformation to tridiagonal form was $A_1 = PAP^T$, then the approximative eigenvectors are the columns of $(\Omega P)^T$.

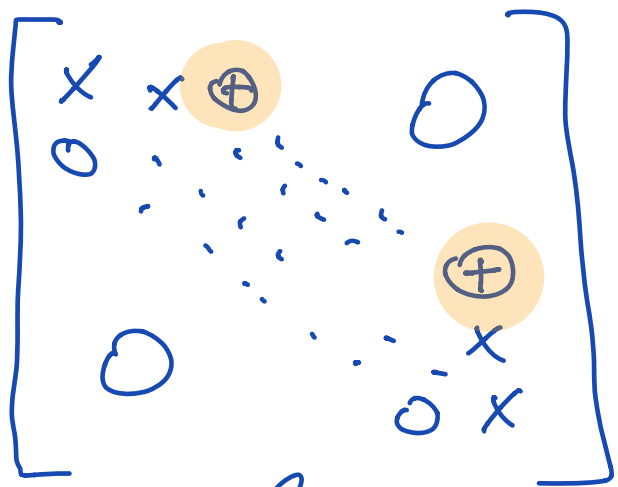
Why is each iteration of the QR algorithm only $O(n^2)$?

Claim: Symmetric tridiagonal matrices remain tridiagonal in the QR algorithm.

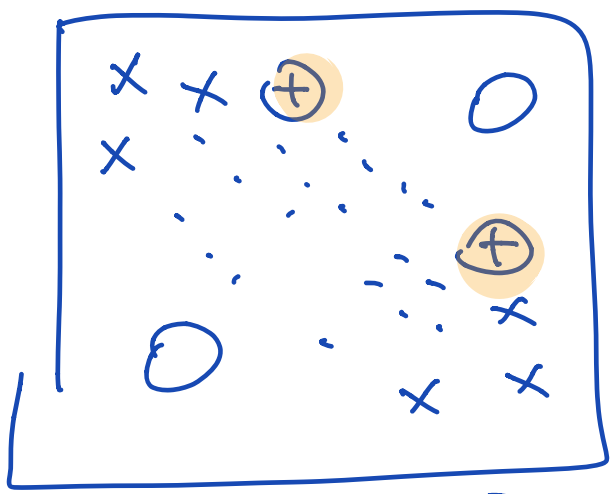


$A = QR$

$n-1$ Givens rotations



$R$



$A' = RQ = Q^T A Q$

Since $A'$ is symmetric, all the ⊕ elements must be zero.

all steps are tridiagonal, and each QR step requires $O(n^2)$ flops.