

연세대학교 통계 데이터 사이언스 학회 ESC 23-2 SUMMER WEEK4

Eigenvalue Problem

[ESC 방학세션 3조] 김채성 노희준 도현수 조준태



Contents

Part I.

1. Overview of Eigenvalue problems
2. Computing a single eigenvalue
3. Basic QR iteration

Part II.

4. Improvements to QR iteration
5. Other methods for eigenvalue problem and connection with SVD



1. Overview of Eigenvalue problems

Basic concepts for eigenvalue problems

Eigenvalues and Eigenvectors

정방행렬 $A \in C^{m \times m}$ 에 대해, $Ax = \lambda x$ 가 성립하면

$x \in C^m$ 는 A 의 eigenvector(고유벡터), $\lambda \in C$ 는 eigenvalue(고유값)라고 한다.

선형변환 A 를 eigenspace S 내에서 단순 scalar multiplication으로 다룬다.

많은 문제들을 scalar problems로 단순화시켜준다는 점에서 매우 유용하다.



Basic concepts for eigenvalue problems

Similarity Transformation

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

(정의) If $X \in C^{m \times m}$ is nonsingular, then the map $A \rightarrow B = X^{-1}AX$ is called **similarity transformation** of A.

- Any similarity transformation is a change of basis operation.
- X^{-1} : X 가 nonsingular이면, I column vector들의 좌표계에서 X 의 column vector들의 좌표계로 변환을 의미한다.
- “A와 B가 닮았다”: A와 B가 동일한 선형변환으로 작용하는 어떤 좌표계(X)가 존재한다.



Basic concepts for eigenvalue problems

Similarity Transformation

(Theorem) If X is nonsingular,

then A and $X^{-1}AX$ have the same characteristic polynomial, eigenvalues, and algebraic and geometric multiplicities.

(Proof)

$$p_{X^{-1}AX}(z) = \det(zI - X^{-1}AX) = \det(X^{-1}(zI - A)X) = \det(X^{-1})\det(zI - A)\det(X) = \det(zI - A) = p_A(z)$$

Characteristic polynomial이란?

$A \in C^{m \times m}$ 의 characteristic polynomial: $p_A(z) = \det(zI - A)$

$\lambda : A$ 의 eigenvalue $\Leftrightarrow p_A(\lambda) = 0$

matrix가 real이더라도, 몇몇 eigenvalue는 complex일 수도 있다.

Relationship between eigenspaces of similar matrices

E_λ 가 A 의 eigenspace라면, $X^{-1}E_\lambda$ 는 $X^{-1}AX$ 의 eigenspace이다.



Basic concepts for eigenvalue problems

Diagonalize

Similarity transformation의 특수한 상황

$A = X\Lambda X^{-1}$: A의 domain 내의 어떤 벡터든, X에서는 A 선형변환이 scalar multiplication으로 단순화될 수 있다.

A의 domain 내의 어떤 벡터든, eigenvector들의 좌표계로 표현 가능하다.

(조건) non-defective matrix여야 한다.

defective란? 대수적 중복도가 기하적 중복도보다 큰 경우 해당 eigenvalue를 defective eigenvalue라고 한다.

Defective eigenvalue가 하나라도 존재할 경우 해당 matrix는 non-diagonalizable이다.

(참고) 대수적 중복도, 기하적 중복도

대수적 중복도: the number of times that polynomial vanishes at that given root λ_j

기하적 중복도: the maximum number of linearly independent eigenvectors of E_λ

- 대수적 중복도는 항상 기하적 중복도보다 크거나 같다.



Basic concepts for eigenvalue problems

Unitary Diagonalization

$A = Q\Lambda Q^*$ 를 만족하는 unitary matrix Q 가 존재할 때, A 를 unitarily diagonalizable하다고 한다.

If and only if a matrix is **normal** \rightarrow **unitarily diagonalizable** (normal matrix: $AA^H = A^H A$)

Hermitian, skew-hermitian, unitary matrix 등이 unitarily diagonalizable하다.

If and only if a matrix is **symmetric** \rightarrow **orthogonally diagonalizable**

[참고]

(real) symmetric ($A = A^T$) \rightarrow (complex) hermitian ($A = A^H$)

(real) orthogonal ($A^T = A^{-1}$) \rightarrow (complex) unitary ($A^H = A^{-1}$)

skew-hermitian: $A = -A^H$



Basic concepts for eigenvalue problems

Unitary Diagonalization

[Example] Unitary diagonalization of Hermitian matrix

$$\begin{pmatrix} 0 & 2i & 0 \\ -2i & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & i & -i \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -2 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 \\ -i & \frac{1}{2} & 0 \\ \frac{i}{2} & \frac{1}{2} & 0 \end{pmatrix}$$

[Example] Unitary diagonalization of skew-Hermitian matrix

$$\begin{pmatrix} 0 & 2i & 0 \\ 2i & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & -1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2i & 0 \\ 0 & 0 & -2i \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & \frac{1}{2} & 0 \end{pmatrix}$$



Schur Factorization

Schur Factorization

A의 슈어분해

$$A = QTQ^*, \quad Q^*AQ = T$$

Q: unitary, T: upper-triangular, A와 T는 similar(닮음)이기 때문에, A의 고유값들은 T의 대각 성분에 반드시 나타난다.

Every square matrix has a Schur factorization.

Schur Factorization

Proof. Every square matrix has a Schur factorization.

Proof We use induction on m . For $m = 1$ we have

$$A = (a_{11}), \quad Q = (1), \quad T = (a_{11}),$$

and the claim is clearly true.

For $m \geq 2$ we assume \mathbf{x} is a normalized eigenvector of A , i.e., $\|\mathbf{x}\|_2 = 1$. Then we form

$$U = \begin{bmatrix} \mathbf{x} & \hat{U} \end{bmatrix} \in \mathbb{C}^{m \times m}$$

to be unitary by augmenting the first column, \mathbf{x} , by appropriate columns in \hat{U} . This gives us

$$U^*AU = \begin{bmatrix} \mathbf{x}^* \\ \hat{U}^* \end{bmatrix} A \begin{bmatrix} \mathbf{x} & \hat{U} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^*A\mathbf{x} & \mathbf{x}^*A\hat{U} \\ \hat{U}A\mathbf{x} & \hat{U}^*A\hat{U} \end{bmatrix}.$$

Since \mathbf{x} is an eigenvector of A we have $A\mathbf{x} = \lambda\mathbf{x}$, and after multiplication by \mathbf{x}^*

$$\mathbf{x}^*A\mathbf{x} = \lambda \underbrace{\mathbf{x}^*\mathbf{x}}_{=\|\mathbf{x}\|_2^2=1} = \lambda.$$

Similarly,

$$\hat{U}^*A\mathbf{x} = \hat{U}^*(\lambda\mathbf{x}) = \lambda\hat{U}^*\mathbf{x} = \mathbf{0}$$

since $\hat{U}^*\mathbf{x} = \mathbf{0}$ because U is unitary.

Therefore, U^*AU simplifies to

$$U^*AU = \begin{bmatrix} \lambda & \mathbf{b}^* \\ \mathbf{0} & C \end{bmatrix}, \quad \mathbf{b}^* = \mathbf{x}^*A\hat{U}, \quad C = \hat{U}^*A\hat{U}$$

Figure 1

수학적 귀납법 이용

- 1 $\times 1$ 에서 Schur factorization 존재 확인.
- 임의의 $A \in \mathbb{C}^{m \times m}$ 를 $(1+m-1) \times (1+m-1)$ 로 분해.
- $(m-1) \times (m-1)$ 에서 Schur factorization 존재 가정.
- 가정 하에 임의의 $A \in \mathbb{C}^{m \times m}$ 가 Schur factorization 존재함을 확인.
⇒ Every square matrix has a Schur factorization.

Schur Factorization

Proof. Every square matrix has a Schur factorization.

$C \in C^{(m-1) \times (m-1)}$ 가 $C = V\hat{T}V^*$ 로 Schur factorization 존재 가정.

To finish the proof we can define

$$Q = U \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & V \end{bmatrix}$$

and observe that

$$\begin{aligned} Q^*AQ &= \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & V^* \end{bmatrix} \underbrace{\begin{bmatrix} U^*AU \\ \lambda & \mathbf{b}^* \end{bmatrix}}_{= \begin{bmatrix} \lambda & \mathbf{b}^* \\ \mathbf{0} & C \end{bmatrix}} \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & V \end{bmatrix} \\ &= \begin{bmatrix} \lambda & \mathbf{b}^*V \\ \mathbf{0} & V^*CV \end{bmatrix}. \end{aligned}$$

This last block matrix, however, is the desired upper triangular matrix T with the eigenvalues of A on its diagonal since $V^*CV = \hat{T}$ and the induction hypothesis ensures \hat{T} already has the desired properties. ■

Figure 2

수학적 귀납법 이용

- 1 $\times 1$ 에서 Schur factorization 존재 확인.
- 임의의 $A \in C^{m \times m}$ 를 $(1 + m - 1) \times (1 + m - 1)$ 로 분해.
- $(m - 1) \times (m - 1)$ 에서 Schur factorization 존재 가정.
- 가정 하에 임의의 $A \in C^{m \times m}$ 가 Schur factorization 존재함을 확인.
→ Every square matrix has a Schur factorization.

몇 단계로 나눠서 해당 품을 만들어가는게 아닌,
수학적 귀납법을 이용해 Schur factorization 존재를 증명

→ 정해진 특정 단계만에 Schur factorization이 되는 것이 아님.
이후 infinite method로의 암시!



Schur Factorization

Eigenvalue-Revealing Factorizations

- A diagonalization(대각화) $A = X\Lambda X^{-1}$ exists if and only if A is nondefective(대수적 중복도=기하적 중복도).
- A unitary diagonalization $A = Q\Lambda Q^*$ exists if and only if A is normal($AA^H = A^HA$).
- A unitary triangularization(Schur Factorization) $A = QTQ^*$ always exists.

=> Schur factorization이 가장 범용적이고, 조건이 맞는 경우에 diagonalization이나 unitary diagonalization을 쓰면 된다.

Overview of Eigenvalue Algorithms

Eigenvalue algorithms - Two phases

Characteristic polynomial을 이용한 정확하고 일반적인 방법들로는 eigenvalue를 구하기 힘들기 때문에,
여러번 반복해서 점차 수렴해나가는 방법을 이용할 것.

목표: 일반적인 matrix를 eigenvalue가 드러나는 structured form(diagonalization, Schur factorization)으로 만든다.
cost를 줄이기 위해서 보통 두 단계로 나누어 eigenvalue problem을 해결한다.

과정:

- 1) Reduction to Hessenberg form
- 2) Convergence to structured form using iterative process

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \underset{A \neq A^*}{\underset{\text{Phase 1}}{\longrightarrow}} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \underset{\text{Phase 2}}{\longrightarrow} \begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ & & & & \times \end{bmatrix}.$$

Figure 3

Overview of Eigenvalue Algorithms

Eigenvalue algorithms - Two phases

이 과정의 목표는 schur factorization 또는 diagonalization을 통해 행렬을 eigenvalue가 드러나는 어떤 행렬로 바꿔주는 것이다. eigenvalue를 유지해야 하므로, similarity transformation을 해야한다.

$$A = QTQ^* \rightarrow T = Q^*AQ = Q_j^* \dots Q_2^*AQ_1 \dots Q_j$$

As $j \rightarrow \infty$, it converges to an upper-triangular matrix T

(Householder triangularization, Gram-Schmidt 같은 방법들과 달리, 이 연산들은 몇번 반복해서 적용해야하는지 정해져있지 않다)

Overview of Eigenvalue Algorithms

Eigenvalue algorithms - Two phases (A is not Hermitian)

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \begin{matrix} \\ \xrightarrow{\text{Phase 1}} \\ \end{matrix} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \begin{matrix} \\ \xrightarrow{\text{Phase 2}} \\ \end{matrix} \begin{bmatrix} x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \\ & & & & x \end{bmatrix}.$$

$A \neq A^*$

H

T

Figure 3

Phase 1: Reduction to upper Hessenberg form (모든 정방행렬은 Hessenberg form으로 변환 가능)

-> cost: 한 열마다 $O(m^2)$ flops, 총 m 열 => $O(m^3)$ flops

Phase 2: Convergence to upper triangular matrix (Householder reflection을 반복적으로 적용)

-> cost: $O(\epsilon_{\text{machine}})$ 으로 수렴시키는 데에 $O(m)$ 정도 반복, 매 반복마다 $O(m^2)$ flops => $O(m^3)$ flops

두 단계로 나누지 않는다면? 매 반복마다 $O(m^3)$ flops => $O(m^4)$ flops 혹은 그 이상. → 비효율적!

Overview of Eigenvalue Algorithms

Eigenvalue algorithms - Two phases (A is Hermitian)

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{\text{Phase 1}} \begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix} \xrightarrow{\text{Phase 2}} \begin{bmatrix} \times & & & \\ & \times & & \\ & & \times & \\ & & & \times \end{bmatrix}.$$

$A = A^*$

T

D

Figure 4

Phase 1: Reduction to tridiagonal form

-> cost: 한 열마다 $O(m^2)$ flops, 총 m 열 => $O(m^3)$ flops

Phase 2: Convergence to diagonal matrix (Householder reflection을 반복적으로 적용)

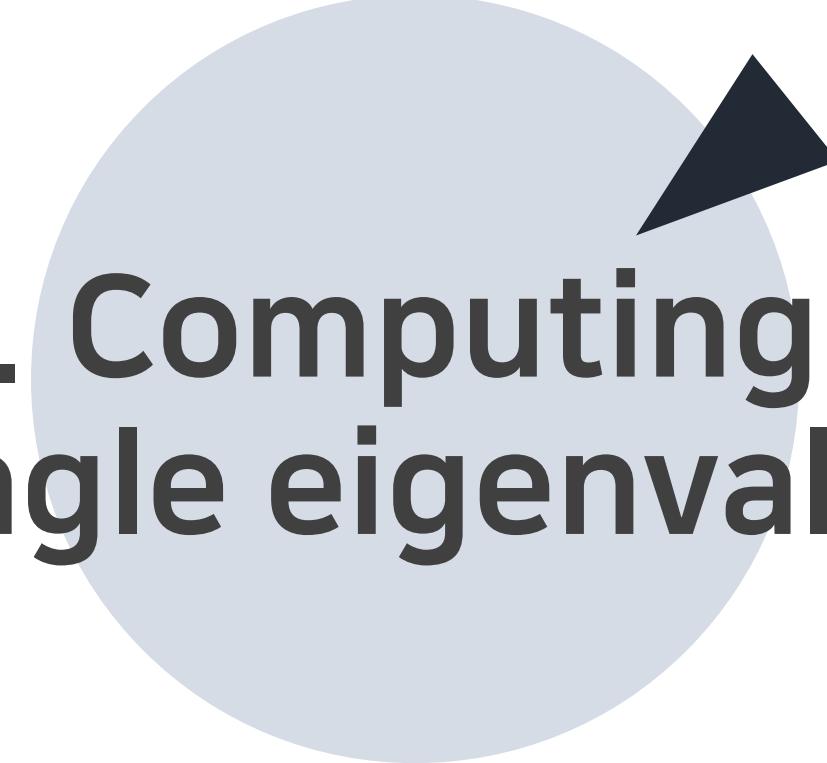
-> cost: 매 반복마다 $O(m)$ flops, m 번 반복 => $O(m^2)$ flops

(참고)

Hermitian matrix란?

$A = \bar{A}^T$. 모든 real symmetric 행렬은 Hermitian이다.

A 가 hermitian이면 Q^*AQ 도 hermitian인데,
모든 hermitian Hessenberg 행렬은 tridiagonal이다.



2. Computing a single eigenvalue

Computing a single eigenvalue

How to compute eigenvalues and eigenvectors

행렬의 정확한 고유값을 구하는 것은 어렵기 때문에, 근사적으로 수렴해가는 방법들을 사용함.

<대표적인 방법>

Single eigenvalue 계산: Power iteration, Inverse iteration, Rayleigh quotient iteration 등.

행렬로 eigenvalue 계산: QR algorithm 및 그 변형들

Computing a single eigenvalue

Rayleigh Quotient

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

만약 x 가 A 의 고유벡터라면, $Ax = \lambda x$ 이고, $r(x) = \lambda$ 로, $r(x)$ 는 A 의 고유값이 된다.

$x\alpha \approx Ax$ 라는 least square problem으로 해석하여, 벡터 x 가 주어졌을 때 $\|Ax - \alpha x\|$ 를 minimize하는 α ,

즉 가장 A 의 고유값과 비슷한 값을 구하는 문제로 해석할 수 있다. => Optimal estimate for the eigenvalue

$y \approx x$ 에 대하여, 단순히 $\frac{\|Ax\|}{\|x\|}$ 로 λ 를 유추하는 방법보다 성능이 좋으며, 그 성능은 다음과 같다.

$$r(y) - r(x) = O(\|y - x\|^2) \text{ as } y \rightarrow x$$

Rayleigh Quotient는 single eigenvalue 계산을 위한 iteration algorithm에서 고유값 계산을 위해 사용된다.

- 1) Power iteration
- 2) Inverse iteration
- 3) Rayleigh Quotient iteration

Computing a single eigenvalue

Power iteration

가정 : A is square diagonalizable matrix이며, Unique Largest eigenvalue를 가질 때. 즉 $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$

$A = X\Lambda X^{-1}$ (행렬 A 의 고유값 분해)

$$A^k = X\Lambda^k X^{-1} = \sum_i \lambda_i^k x_i y_i^T$$

λ_1 이 Unique largest eigenvalue이고, 다른 고유값들에 비해 매우 크므로, 다음과 같은 근사식으로 표현할 수 있다.

$$A^k \approx \lambda_1^k x_1 y_1^T$$

양변에 임의의 벡터 z 를 곱하면

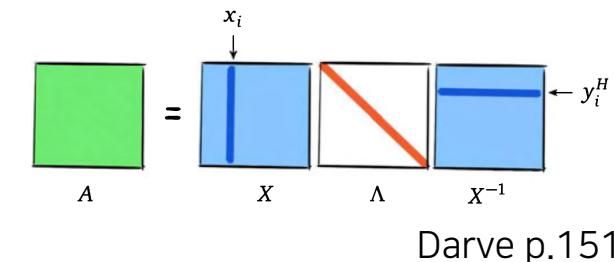
$A^k z \approx \lambda_1^k x_1 y_1^T z = (\lambda_1^k y_1^T z) x_1$ 가 되고, x_1 의 approximation, 즉 고유벡터의 approximation을 다음과 같이 구할 수 있다.

$$\tilde{x}_1 = \frac{A^k z}{\|A^k z\|}$$

이에 따라, Rayleigh Quotient를 이용하여 고유값 λ_1 의 approximation도 다음과 같이 계산할 수 있다.

$$\lambda_1 \approx \tilde{x}_1^T A \tilde{x}_1$$

이처럼 임의의 벡터에 행렬 A 를 여러번 곱하는 Iteration으로 고유벡터와 고유값의 approximation을 구하는 것이 power iteration의 아이디어라고 할 수 있다.



Computing a single eigenvalue

Power iteration

임의의 행렬 A 의 가장 큰 eigenvalue에 상응하는 고유벡터를 구하는 iteration

[Algorithm]

$q^{(0)}$ = some vector with $\|q^{(0)}\| = 1$

for k=1,2,...

$$w = Aq^{(k-1)} \quad (\text{Apply A})$$

$$q^{(k)} = w / \|w\| \quad \text{(Normalize)}$$

$$\lambda^{(k)} = (q^{(k)})^T A q^{(k)} \quad \text{(Rayleigh quotient)}$$

임의의 정규벡터 $g^{(0)}$ 으로 시작해,

행렬 A 를 곱하고 정규화시키는 과정을 반복해 $q^{(k)}$ 를 update하고 A 의 eigenvector로 근사시킨다.

$q^{(k)} \approx q^{(k-1)}$ 가 될 때 iteration을 멈춘다.

Rayleigh quotient $r(q^{(k)})$ 로 eigenvalue를 구한다.

$$q^{(k)} \rightarrow x_1, \lambda^{(k)} \rightarrow \lambda_1$$

- Convergence

$$\|q^{(k)} - x_1\| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right) \text{ (linear), } |\lambda^{(k)} - \lambda_1| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right) \text{ (quadratic)}$$



Computing a single eigenvalue

power iteration의 convergence

벡터 $q^{(0)}$ 에서 iteration을 시작한다고 해보자.

$q^{(0)} = a_1x_1 + \dots + a_nx_n$ 의 선형결합으로 표현할 수 있다.

$a_1 \neq 0$, $v^{(0)}$ 는 x_1 에 orthogonal하지 않다고 가정

$$A^k q^{(0)} = \sum_i a_i A^k x_i = \sum_i a_i \lambda_i^k x_i = a_1 \lambda_1^k x_i = a_1 \lambda_1^k (x_1 + \frac{a_2}{a_1} (\frac{\lambda_2}{\lambda_1})^k x_2 + \dots + \frac{a_n}{a_1} (\frac{\lambda_n}{\lambda_1})^k x_n)$$

이에 따라, $\|A^k q^{(0)}\| = |a_1 \lambda_1^k| (1 + O(|\frac{\lambda_2}{\lambda_1}|^k))$, $\|(a_1 \lambda_1^k)^{-1} A^k v^{(0)} - x_1\| = O(|\frac{\lambda_2}{\lambda_1}|^k)$

$$\Rightarrow \|q^{(k)} - x_1\| = O(|\frac{\lambda_2}{\lambda_1}|^k) \text{ (linear)}$$

Using Rayleigh quotient, $|\lambda^{(k)} - \lambda_1| = O(|\frac{\lambda_2}{\lambda_1}|^{2k})$ (Quadratic). 전단계보다 제곱으로 빠르게 수렴한다.

Computing a single eigenvalue

Power iteration

하지만 Power iteration에는 다음과 같은 한계가 있다.

- $|\frac{\lambda_2}{\lambda_1}|$ 가 충분히 작지 않다면 굉장히 느린 수렴 속도.
- 오직 largest eigenvalue만 찾아낼 수 있다.

이러한 문제점을 개선한 알고리즘: Inverse iteration, Rayleigh iteration

Computing a single eigenvalue

Inverse iteration

임의의 행렬 A (real, symmetric)의 고유값 λ_i 의 추정치 μ 를 사용해 고유값과 고유벡터를 구하는 iteration

Power iteration과 형태는 비슷하나, 행렬 A 대신 $(A - \mu I)^{-1}$ 를 계산에 사용한다.

A 의 고유값: λ_i $(A - \mu I)^{-1}$ 의 고유값: $(\lambda_i - \mu)^{-1}$

A 와 $(A - \mu I)^{-1}$ 의 고유벡터는 동일하다. $(Av = \lambda v \Leftrightarrow (A - \mu I)v = \lambda v - \mu v \Leftrightarrow (\lambda - \mu)^{-1}v = (A - \mu I)^{-1}v)$

-> 이 특성을 활용!

만약 μ 가 λ_i 에 가깝다면, $(\lambda_i - \mu)^{-1}$ 은 굉장히 클 것이고, $(A - \mu I)^{-1}$ 를 곱해나가는 power iteration이 매우 빠르게 수렴할 것이다.

power iteration은 largest eigenvalue에만 적용될 수 있지만, inverse iteration은 μ 에 가까운 어떤 eigenvalue에 구할 수 있다.

Unique largest eigenvalue가 없어도 적당한 λ_i 에 대한 추정값 μ 를 갖고 있다면 eigenvalue를 찾아낼 수 있다.

$|\frac{\lambda_2}{\lambda_1}|$ 가 작아도 $|\frac{1}{\lambda_i - \mu}|$ 를 크게 만들어서 더 좋은 성능을 기대할 수 있다.

Computing a single eigenvalue

Inverse iteration

임의의 행렬 A (real, symmetric)의 고유값 λ_i 에 근사하는 상수 μ 를 사용해 고유값과 고유벡터를 구하는 iteration

[Algorithm]

$q^{(0)}$ = some vector with $\|q^{(0)}\| = 1$

for $k=1,2,\dots$

$$w = (A - \mu I)^{-1}v^{(k-1)} \quad (\text{Apply } (A - \mu I)^{-1})$$

$$q^{(k)} = w/\|w\| \quad (\text{Normalize})$$

$$\lambda^{(k)} = (q^{(k)})^T A q^{(k)} \quad (\text{Rayleigh quotient})$$

$q^{(k)}$ 는 μ 에 가장 가까운 고유값으로, $\lambda^{(k)}$ 는 그에 상응하는 고유벡터로 근사하게 된다.

- Convergence

λ_i : μ 에 가장 가까운 고유값, λ_j : μ 에 2번째로 가까운 고유값, 즉 $|\mu - \lambda_i| < |\mu - \lambda_j| \leq |\mu - \lambda_k|$ ($k \neq i$)라고 하자.

$$\|q^{(k)} - x_i\| = O\left(\left|\frac{\lambda_i - \mu}{\lambda_j - \mu}\right|^k\right) \text{ (linear)}, \quad |\lambda_i^{(k)} - \lambda_i| = O\left(\left|\frac{\lambda_i - \mu}{\lambda_j - \mu}\right|^{2k}\right) \text{ (Quadratic)}$$

Computing a single eigenvalue

Rayleigh Quotient iteration

Inverse iteration과 원리는 같으나, 각 iteration에서 μ 대신 rayleigh quotient를 통해 update한 $\lambda^{(k-1)}$ 를 사용하여 Convergence 속도를 향상.

가정: A is a real symmetric matrix

[Algorithm]

$q^{(0)}$ = some vector with $\|q^{(0)}\| = 1$

$\lambda^{(0)} = (q^{(0)})^T A q^{(0)}$ corresponding Rayleigh quotient

for $k=1,2,\dots$

$$w = (A - \lambda^{(k-1)}I)^{-1}q^{(k-1)} \quad (\text{Apply } (A - \lambda^{(k-1)}I)^{-1})$$

$$q^{(k)} = w/\|w\| \quad (\text{Normalize})$$

$$\lambda^{(k)} = (q^{(k)})^T A q^{(k)} \quad (\text{Rayleigh quotient})$$

- Convergence

$$\|q^{(k+1)} - x_i\| = O(\|q^{(k)} - x_i\|^3) = O(\epsilon^3) \text{ (cubic), } |\lambda^{(k+1)} - \lambda_i| = O(|\lambda^{(k)} - \lambda_i|^3) \text{ (cubic)}$$

=> 다른 Iteration들보다 빠르다!



Computing a single eigenvalue

Rayleigh Quotient iteration

[Example]

Consider the matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 1 \\ 3 & 2 & 1 \end{bmatrix}$$

for which the exact eigenvalues are $\lambda_1 = 3 + \sqrt{5}$, $\lambda_2 = 3 - \sqrt{5}$ and $\lambda_3 = -2$, with corresponding eigenvectors

$$v_1 = \begin{bmatrix} 1 \\ \varphi - 1 \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} 1 \\ -\varphi \\ 1 \end{bmatrix} \text{ and } v_3 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

(where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio).

The largest eigenvalue is $\lambda_1 \approx 5.2361$ and corresponds to any eigenvector proportional to $v_1 \approx \begin{bmatrix} 1 \\ 0.6180 \\ 1 \end{bmatrix}$.

Figure 5

We begin with an initial eigenvalue guess of

$$b_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \mu_0 = 200.$$

Then, the first iteration yields

$$b_1 \approx \begin{bmatrix} -0.57927 \\ -0.57348 \\ -0.57927 \end{bmatrix}, \mu_1 \approx 5.3355$$

the second iteration,

$$b_2 \approx \begin{bmatrix} 0.64676 \\ 0.40422 \\ 0.64676 \end{bmatrix}, \mu_2 \approx 5.2418$$

and the third,

$$b_3 \approx \begin{bmatrix} -0.64793 \\ -0.40045 \\ -0.64793 \end{bmatrix}, \mu_3 \approx 5.2361$$

from which the cubic convergence is evident.

Figure 6



Computing a single eigenvalue

Convergence - Power iteration

$$\|q^{(k)} - x_1\| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right) \text{ (linear)}, \quad |\lambda^{(k)} - \lambda_1| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right) \text{ (Quadratic). [p.23 참고]}$$

Convergence - Inverse iteration

Power iteration과 동일한 원리로 구하되, λ 가 $\lambda - \mu$ 로 대체된다.

λ_i : μ 에 가장 가까운 고유값, λ_j : μ 에 2번째로 가까운 고유값, 즉 $|\mu - \lambda_i| < |\mu - \lambda_j| \leq |\mu - \lambda_k|$ ($k \neq i$)라고 하자.

$$\|q^{(k)} - x_i\| = O\left(\left|\frac{\lambda_i - \mu}{\lambda_j - \mu}\right|^k\right) \text{ (linear)}$$

$$|\lambda_i^{(k)} - \lambda_i| = O\left(\left|\frac{\lambda_i - \mu}{\lambda_j - \mu}\right|^{2k}\right) \text{ (Quadratic)}$$

μ 가 λ_i 에 충분히 가깝다면 $\left|\frac{\lambda_i - \mu}{\lambda_j - \mu}\right|$ 가 매우 작아져서 향상된 수렴 속도를 누릴 수 있다.



Computing a single eigenvalue

Convergence - Rayleigh Quotient iteration

$\|q^{(k)} - x_i\| = O(\epsilon)$ 이라고 하면,

$\|q^{(k+1)} - x_i\| = O(\epsilon |\lambda_i - \mu|)$

$\mu = r(q^{(k)})$ 이고, $\|q^{(k)} - x_i\| \leq \epsilon_0$ 이기 때문에,

$|\lambda_i - \mu| = O(\epsilon^2)$ 가 되고, 이를 대입하면

$\|q^{(k+1)} - x_i\| = O(\epsilon^3)$ (cubic)

즉, 전단계보다 세제곱 빠르게 수렴한다.

Quadratic이었던 기존 방법(Power, inverse)보다 성능이 더 좋다.

$$\begin{array}{ccc} \|v^{(k)} - (\pm q_J)\| & & |\lambda^{(k)} - \lambda_J| \\ \epsilon & \rightarrow & O(\epsilon^2) \end{array}$$

$$\begin{array}{ccc} & \downarrow & \swarrow \\ O(\epsilon^3) & \rightarrow & O(\epsilon^6) \end{array}$$

$$\begin{array}{ccc} & \downarrow & \swarrow \\ O(\epsilon^9) & \rightarrow & O(\epsilon^{18}) \end{array}$$

\vdots \vdots

Figure 7

3. Basic QR iteration

Orthogonal iteration

QR iteration은 앞서 살펴본 single eigenvalue computation들과 달리, 행렬의 모든 eigenvalue들을 한번에 구할 수 있다.

[3. Basic QR iteration]에서는, 행렬 A 가 고유값 $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|$ (모든 고유값이 real, distinct)으로 diagonalizable하다고 가정한다.

Orthogonal iteration (for $r = 2$, r : 고유값 개수)

[Algorithm]

임의의 직교 벡터 q_1, q_2 설정

while not converged:

$$q_1 \leftarrow Aq_1, q_2 \leftarrow Aq_2$$

q_2 를 q_1 에 orthogonal인 공간으로 project

q_1, q_2 를 normalize

$$q_1^{(k)} = \frac{A^k q_1^{(0)}}{\|A^k q_1^{(0)}\|}, q_1 \rightarrow x_1 : \text{power iteration of } A$$

$$q_2^{(k)} \approx (I - x_1 x_1^T) A q_2^{(k-1)}, q_2 \rightarrow (I - x_1 x_1^T) x_2 : \text{power iteration of } (I - x_1 x_1^T) A$$

$((I - x_1 x_1^T)$ 는 projector ; orthogonal to x_1

$\Rightarrow \text{span}\{q_1, q_2\}$ converges to $\text{span}\{x_1, x_2\}$

(x_1, x_2 는 A 의 고유벡터)



Orthogonal iteration

Orthogonal iteration

How to find eigenvalues λ_1, λ_2 ?

Compute $Q_k^T A Q_k$ where $Q_k = [q_1 \ q_2]$. => Upper Triangular matrix. (Similarity Transformation이 아니지만, eigenvalue도 보존됨.)

$Q_k^T A Q_k$: Schur factorization과 동일한 품. But, Q_k is not Unitary matrix.

[lower-left entry가 0 되는 이유]

$$q_2^T A q_1 \approx \lambda_1 q_2^T q_1 = 0$$

[대각성분들이 eigenvalue인 이유]

$\text{span}\{q_1, q_2\} \approx \text{span}\{x_1, x_2\}$ 이기 때문에 $Q_k v_i \approx x_i$ 를 만족하는 벡터 v_i 가 있을 것이다

$Q_k^T A Q_k v_i \approx Q_k^T A x_i = \lambda_i Q_k^T x_i \approx \lambda_i Q_k^T Q_k v_i = \lambda_i v_i$ 이기 때문에, λ_1, λ_2 는 고유값, v_1, v_2 는 $Q_k^T A Q_k$ 이에 상응하는 고유벡터가 된다.

Upper triangle matrix의 경우, 그 고유값이 언제나 대각성분에 나타나나므로, $Q_k^T A Q_k$ 의 대각 성분들은 고유값이다.

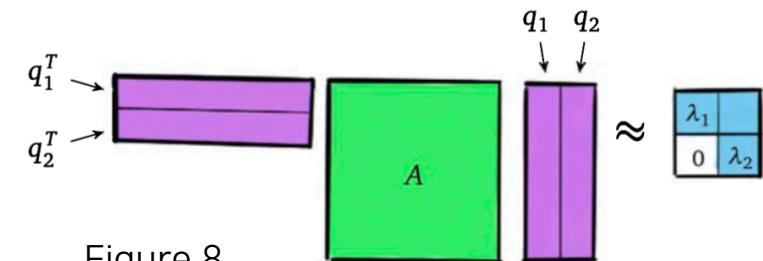


Figure 8



Orthogonal iteration

Orthogonal iteration (for general r)

[Algorithm]

for $k=1, \dots, n$

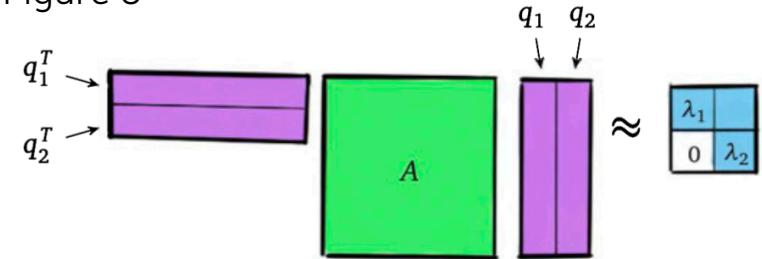
$$Q_k = A Q_k$$

임의의 Q_1 로 시작

$$Q_{k+1} R_{k+1} = Q_k$$

Orthogonalize the columns using QR (Classical Gram-Schmidt)

Figure 8



Define $[x_1 | \dots | x_r] = X_r = Q^x R^x$. QR분해의 결과인 $Q^x \in R^{n \times r}$ 는 결국 우리가 찾던 orthogonal set인 $Q_k \rightarrow Q^x$ as $k \rightarrow \infty$.

$$AX_r = X_r \Lambda_r \quad X_r \in C^{n \times r}, \Lambda_r \in R^{r \times r}$$

$$AX_r = A Q^x R^x = Q^x R^x \Lambda_r, \quad A Q^x = Q^x R^x \Lambda_r R^{x-1}, \rightarrow (Q^x)^T A Q^x = R^x \Lambda_r (R^x)^{-1}$$

$R^x, \Lambda_r, (R^x)^{-1}$ 은 모두 Upper Triangular matrix. 따라서 $(Q^x)^T A Q^x$ 는 Upper Triangular matrix이며, 고유값 Λ_r

$+ \text{span}\{Q^x\} \approx \text{span}\{x_1, \dots, x_r\}$ 이므로, $r=2$ 에서와 같은 이유로 $(Q^x)^T A Q^x$ 의 대각 성분들은 고유값 $\lambda_1, \dots, \lambda_r$ 이다.

=> 어떤 r 에 대해서든지, $Q_k^T A Q_k$ 는 $(Q^x)^T A Q^x$ 로 수렴하고, 이는 A 의 고유값 $\lambda_1, \dots, \lambda_r$ 을 대각성분으로 가지는 상삼각행렬이다.



Orthogonal iteration

Orthogonal iteration (for $r = n$)

A is diagonalizable, $A = X\Lambda X^{-1}$

When $r = n$, $[x_1 | \dots | x_n] = X = Q^x R^x$

Using $X = Q^x R^x$, $(Q^x)^T A Q^x = R^x \Lambda_r (R^x)^{-1}$ and also Upper Triangular matrix.

For $r = n$, $(Q^x)^T A Q^x = T$ is the Schur factorization.

$\rightarrow Q^x$: Unitary matrix, similarity transformation, upper triangular matrix T with same eigenvalues.

As $k \rightarrow \infty$, $(Q^x)^T A Q^x = T$ converges to Schur factorization.



QR iteration

QR iteration이란?

A 에 대한 슈어 분해 $T = Q^T A Q$ 를 구하는 iteration

Orthogonal iteration에서는 Q 를 update했는데, QR iteration에서는 T 를 바로 구하고자 함.

[Pure QR Algorithm]

$T_k = Q_k^T A Q_k$ (kth iterate of orthogonal iteration)가 있을 때, 다음 과정으로 T 를 update한다.

1. $T_k = U_{k+1} R_{k+1}$ 로 QR 분해
2. $T_{k+1} = R_{k+1} U_{k+1}$ 로 update

충분히 많이 update하고 나면 상삼각행렬 T 의 대각 성분들은 고유값으로 수렴한다.

QR iteration

QR iteration이란?

$$T_k = U_{k+1}R_{k+1}$$

$$T_{k+1} = R_{k+1}U_{k+1}$$
가 되는 이유

정의에 따라, $T_k = Q_k^T A Q_k$, $T_{k+1} = Q_{k+1}^T A Q_{k+1}$

orthogonal iteration의 정의에 따라, $Q_{k+1}R_{k+1} = A Q_k$. 이에 따라, $R_{k+1}Q_k^T = Q_{k+1}^T A$

이를 이용하여 계산하면

$$T_k = Q_k^T (Q_{k+1}R_{k+1})$$

$$T_{k+1} = (R_{k+1}Q_k^T)Q_{k+1}$$

$U_{k+1} = Q_k^T Q_{k+1}$ 라고 정의하자. 이때, U_{k+1} 은 orthogonal이다.

따라서,

$T_k = U_{k+1}R_{k+1}$, $T_{k+1} = R_{k+1}U_{k+1}$ 임을 알 수 있다.

$$T_k = Q_k^T A Q_k = Q_k^T (Q_{k+1}R_{k+1})$$



QR iteration

QR iteration이란?

[Pure QR Algorithm - Symmetric case]

A가 대칭행렬

$T_k = Q_k^T A Q_k$ (kth iterate of orthogonal iteration)가 대칭행렬인 경우, 대각행렬로 수렴한다(대칭+상삼각).

1. $T_k = U_{k+1} R_{k+1}$ 로 QR 분해

2. $T_{k+1} = R_{k+1} U_{k+1}$ 로 update

충분히 많이 update하고 나면 대칭행렬 T의 대각 성분들은 고유값으로 수렴한다.

[Example] Let $A_0 = \begin{bmatrix} 7 & 2 \\ 2 & 4 \end{bmatrix}$

$$\text{QR factorization 계산 } A_0 = Q_1 R_1 = \begin{bmatrix} .962 & -.275 \\ .275 & .962 \end{bmatrix} \begin{bmatrix} 7.28 & 3.02 \\ 0 & 3.30 \end{bmatrix}$$

$$\text{Reverse } A_1 = R_1 Q_1 = \begin{bmatrix} 7.83 & .906 \\ .906 & 3.17 \end{bmatrix}$$

비대각성분들은 처음보다 작아지고, 대각성분들은 고유값(8,3)에 더 가까워졌다.
A가 대각 행렬이 될 때까지 반복하면 대각 성분들은 고유값으로 근사한다.



4. Improvements to QR iteration

Improvements to QR iteration

QR iteration의 단점

효율성 문제: cost가 $O(n^3)$ 인 QR 분해를 $O(n)$ 번 반복할 경우, $O(n^4)$ flops로, 계산 비용이 매우 크다

수렴 문제: 수렴이 고유값들의 분포에 크게 영향 받는다

ex) 두 고유값의 크기가 같을 경우, 정상적으로 수렴하지 않을 수 있다

<문제 개선 방법>

1. QR 분해의 계산 비용을 줄이기 위해 A 를 upper Hessenberg 형태로 만들자
2. 수렴 속도를 높이기 위해 QR algorithm의 shifted version을 활용하자

Improvements to QR iteration

-Starting with an upper Hessenberg matrix

Reducing the cost of QR

전제: A 는 대각화 가능하고, $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|$ (모든 고유값이 real, distinct)

QR iteration의 가장 큰 장애물은 QR의 cost인 만큼, A 를 QR이 더 빠르게 계산될 수 있는 형태(Hessenberg matrix)로 바꾸자!

1. $H = Q^T A Q$ 를 통해 A 를 upper Hessenberg matrix로 만들어준다
2. H 에 QR iteration을 적용하여 Schur 분해를 실시한다

$$T = (Q')^T H (Q') = (QQ')^T A (QQ')$$

Householder Reduction

Householder Reduction to Hessenberg form

슈어 분해 $A = QTQ^*$ 에서, 상삼각행렬 T를 구하기 전에 A에 대한 transformation을 적용하여 Hessenberg form으로 변형

Householder reflector를 활용해 Hessenberg matrix H 를 구한다. A 가 $m \times m$ 행렬일 때,

$$H = Q^* A Q = Q_{m-2}^* \dots Q_1^* A Q_1 \dots Q_{m-2}$$

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{Q_1^*} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \xrightarrow{\cdot Q_1} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}$$

Figure 9

성분들을 0으로 바꾼 후, Similarity transformation을 해야하기 때문에 O_1 을 곱함

$$\underbrace{Q_{m-2}^* \cdots Q_2^* Q_1^*}_Q^* A \underbrace{Q_1 Q_2 \cdots Q_{m-2}}_Q = H$$

Figure 10



Householder Reduction

Householder Reduction to Hessenberg form

- Householder matrix Q의 형태

$Q_k = \begin{pmatrix} I_k & 0 \\ 0 & P_k \end{pmatrix}$, P_k 는 $A_{k+1:m,k}$ (0으로 바꿀 열의 대각원소를 제외한 성분들의 벡터)를 $\|A_{k+1:m,k}\|e_1$ 로 매팅하는 Householder reflection

+ 상삼각행렬이 아닌 Hessenberg form을 만드는 이유?

대각성분 아래의 성분들을 0으로 만드는 Householder reflector를 적용하여 한번에 상삼각행렬 T를 구하려고 하면,
고유값 보존을 위해 similarity transformation 시 다시 0이 아닌 것들로 되돌아 올 수 있음

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{Q_1^*} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix} \xrightarrow{\cdot Q_1} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} = Q_1^* A Q_1$$

Figure 9

Householder Reduction

Householder Reduction to Tridiagonal form

- 만약 행렬 A 가 Hermitian matrix라면, A 는 Hessenberg가 아닌 tridiagonal form으로 reduced.

Hermitian matrix란? $A = \bar{A}^T$. 모든 real symmetric 행렬은 Hermitian 이다.

A 가 hermitian이면 Q^*AQ 도 hermitian인데, 모든 hermitian Hessenberg 행렬은 tridiagonal이다.

Example] hermitian matrix

$$\begin{pmatrix} 2 & 2+i & 4 \\ 2-i & 3 & i \\ 4 & -i & 1 \end{pmatrix}$$

Example] tridiagonal matrix

$$\begin{pmatrix} x & x & & & \\ x & x & x & & \\ x & x & x & & \\ & x & x & x & \\ & x & x & x & \end{pmatrix}$$

Improvements to QR iteration

-Starting with an upper Hessenberg matrix

Computing H

'Householder reflections를 반복적으로' 사용하자!

$$H = Q_{n-2}^T \dots Q_1^T A Q_1 \dots Q_{n-2} \rightarrow H \text{는 upper Hessenberg 행렬}$$
$$\rightarrow (n-2) \text{ Householder Transformations} * O(n^2) = O(n^3) \text{ flops}$$

Step 1: Find Q_1 to put zeros in the first column

$$\begin{bmatrix} 1 \\ P_1 \end{bmatrix} \begin{bmatrix} v \\ A \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ P_1 \end{bmatrix} \begin{bmatrix} v \\ A \end{bmatrix} \begin{bmatrix} 1 \\ P_1^T \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} 1 & 0 \\ 0 & P_1 \end{bmatrix}$$

$$\dots \Rightarrow \underbrace{\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \dots \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}}_{Q^T} \dots \underbrace{\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \dots \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}}_{Q} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} H$$

Step 2: Find Q_2 to put zeros in the second column

$$\begin{bmatrix} 1 \\ P_2 \end{bmatrix} \begin{bmatrix} v \\ A \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ P_2 \end{bmatrix} \begin{bmatrix} v \\ A \end{bmatrix} \begin{bmatrix} 1 \\ P_2^T \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$



Improvements to QR iteration

-Starting with an upper Hessenberg matrix

Computing H

$$A = \begin{bmatrix} 5 & -2 & 2 \\ 4 & -3 & 4 \\ 3 & -6 & 7 \end{bmatrix}$$

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & P \\ 0 & 0 \end{bmatrix}$$

$$H = Q^T A Q$$

→ A의 고유값들은 유지가 되면서, QR iteration 수렴 속도는 빨라짐.

① Entries of 1st column (not A_{11})

$$x = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

② $v = x + \text{sign}(x_1) \cdot \|x\| \cdot e_1$

$$= \begin{bmatrix} 4 \\ 3 \end{bmatrix} + 5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 9 \\ 3 \end{bmatrix}$$

③ $v^T v = [9 0]$

$$v v^T = \begin{bmatrix} 81 & 27 \\ 27 & 9 \end{bmatrix}$$

④ $I - \frac{2v v^T}{v^T v} = \begin{bmatrix} -0.8 & -0.6 \\ -0.6 & 0.8 \end{bmatrix} = P$

$$\therefore Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.8 & -0.6 \\ 0 & 0.6 & 0.8 \end{bmatrix}$$

$$H = Q^T A Q = \begin{bmatrix} 5 & 0.4 & 2.8 \\ -5 & 0.44 & -8.92 \\ 0 & 0.48 & 4.36 \end{bmatrix}$$



Improvements to QR iteration

-Starting with an upper Hessenberg matrix

QR iteration on H

First step: Find U_k, R_k so that $U_k R_k = T_{k-1}$

Hessenberg 형태인 H 에 QR iteration을 적용하므로 T_{k-1} 은 계속 upper Hessenberg 형태를 띠고,

T_{k-1} 이 upper Hessenberg 형태이므로, Givens rotation G_1^T, \dots, G_{n-1}^T 을 이용해 더 효율적으로 수행할 수 있음을 이용하자!

$$R_k = G_{n-1}^T \cdots G_1^T T_{k-1}$$

→ G_1^T, \dots, G_{n-1}^T 를 곱함으로써 각각 #의 1열, ..., n-1열의 sub-diagonal entry를 0으로 바꿔준다

⇒ R_k 는 upper triangular 형태를 띤다

Improvements to QR iteration

-Starting with an upper Hessenberg matrix

QR iteration on H

Second step: Compute $T_k = R_k U_k$

$$T_k = R_k U_k$$

$$= R_k G_1 \cdots G_{n-1}$$

→ Givens rotation, G_1, \dots, G_{n-1} 를 R_k 의 column에 적용함을 의미한다

⇒ T_k 는 upper Hessenberg 형태로 유지된다

Total cost

$H = Q^T A Q$ 로 upper Hessenberg form H 만들어주기 → $O(n^3)$ flops

QR iteration → $n * O(n^2) = O(n^3)$ flops

⇒ Total cost는 $O(n^3)$ 으로, $O(n^4)$ 에 비해 감소한 모습을 보인다!



Improvements to QR iteration

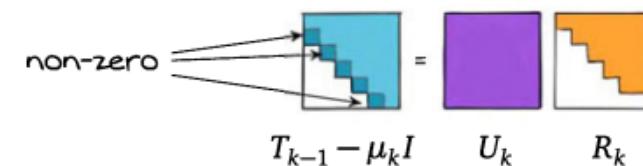
-QR with a shift strategy

Adding a shift

전제: Upper Hessenberg form H ($H = Q^T A Q$)를 초기 행렬로 사용하고, 고유값들이 각각 다른 크기를 가질 필요가 없다

1. $H = Q^T A Q$ 를 통해 A 를 Upper Hessenberg matrix H 로 만들어주고, H 를 초기행렬 T_0 로 설정한다 ($k = 1, 2, \dots$ 에 대하여 반복 진행)
2. Shift μ_k 를 선택한다
3. 주어진 upper Hessenberg matrix T_{k-1} 을 바탕으로 QR 분해를 진행한다

$$T_{k-1} - \mu_k I = U_k R_k$$


$$T_{k-1} - \mu_k I = U_k R_k$$

4. $T_k = R_k U_k + \mu_k I$ 를 통해 RQ 재결합을 진행한다

→ T_k 는 A 와 같은 고유값을 가지므로, A 의 고유값들을 대각 성분으로 가지는 상삼각 행렬로 수렴한다

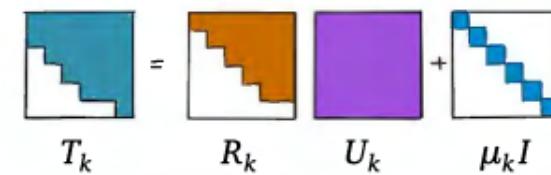

$$T_k = R_k + U_k + \mu_k I$$

Figure 12

*** T_k 의 비대각선 성분들이 0이거나 0에 가까이 수렴할 경우, T_k 를 부분행렬 T_1 과 T_2 로 나눈 후, 각각 QR 알고리즘을 적용하여 고유값과 고유벡터를 효율적으로 구할 수 있다

$$T_k \approx \begin{bmatrix} (T_k)_{11} & (T_k)_{12} \\ 0 & (T_k)_{22} \end{bmatrix}$$

Figure 13



Improvements to QR iteration

-QR with a shift strategy

Choosing the shifts-Simple approach

Simple approach: T_k 의 lower-right entry를 shifts로 고르자!

$$\mu_k = (T_k)_{nn}$$

T_k 는 A 의 고유값을 대각 성분으로 가지는 upper-triangular 행렬로 수렴하므로,

T_k 의 lower-right entry인 $(T_k)_{nn}$ 을 선택하는 것은 가장 작은 고유값을 선택하는 것과 유사하다

이 경우, $T_{k-1} - \mu_k I$ 의 고유값인 $\lambda_i - \mu$ 의 크기 순서가 기존과 거의 같게 나타나므로, 수렴 속도가 향상될 수 있다

Improvements to QR iteration

-QR with a shift strategy

Choosing the shifts-Simple approach

[한계]

A 가 실수행렬이지만, λ_n 은 복소수일 때, shift를 사용해도 효과가 없다

ex)

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

$\rightarrow \lambda = \pm i, |\lambda_1| = |\lambda_2|$ 이므로, normal QR iteration은 효과가 없다

$\rightarrow \mu_k((T_k)_{nn})$ 을 0으로 선택하므로 shifted QR iteration 또한 효과가 없다

\Rightarrow 이러한 문제를 해결하기 위해 T_k 의 마지막 2×2 block에 집중하자!

만약 이 마지막 2×2 block의 고유값들이 실수라면, 작은 값을 shift로 사용할 수 있다

하지만 복소수라면, $(\mu, \bar{\mu})$ 형태의 고유값 쌍이 생겨 둘 중 하나를 임의로 선택해 shift로 사용할 수 있고, 복소 행렬이 만들어지게 된다



Improvements to QR iteration

-QR with a shift strategy

Choosing the shifts-Francis shift

Francis shift: 앞선 idea를 활용하면서, A 의 실수행렬 형태를 유지하고자 할 때 사용되는 방법

: μ 를 shift로 사용한 후, $\bar{\mu}$ 를 shift로 사용하는 'Double shift' 방법을 활용한다

Double Shift: 처음 shift(μ) 사용 시에는 복소 행렬이 되지만, 두번째 shift($\bar{\mu}$) 사용 시 다시 실수 행렬로 돌아온다는 원리!

$$\begin{aligned}T - \mu I &= U_1 R_1, \\T_1 &= R_1 U_1 + \mu I, \\T_1 - \bar{\mu} &= U_2 R_2, \\T_2 &= R_2 U_2 + \bar{\mu} I\end{aligned}$$

→ $U_1 U_2$ 가 실수 행렬이 되어 T_2 도 실수 행렬이 된다

proof

$$s = \mu + \bar{\mu}, t = |\mu|^2 \text{ 일 때,}$$

$$\begin{aligned}(U_1 U_2)(R_2 R_1) &= U_1(T_1 - \bar{\mu} I)R_1 \\&= U_1(R_1 U_1 + (\mu - \bar{\mu})I)R_1 \\&= U_1 R_1(U_1 R_1 + (\mu - \bar{\mu})I) \\&= (T - \mu I)(T - \mu I + (\mu - \bar{\mu})I) \\&= (T - \mu I)(T - \bar{\mu} I) = T^2 - sT + tI\end{aligned}$$

→ $T^2 - sT + tI$ 는 실수 행렬이므로, $U_1 U_2$ 도 실수 행렬이다



Improvements to QR iteration

-QR with a shift strategy

QR iteration with shift

[Example] Let $A_0 = \begin{bmatrix} 7 & 2 \\ 2 & 4 \end{bmatrix}$

QR factorization 계산 $A_0 = Q_1 R_1 = \begin{bmatrix} .962 & -.275 \\ .275 & .962 \end{bmatrix} \begin{bmatrix} 7.28 & 3.02 \\ 0 & 3.30 \end{bmatrix}$

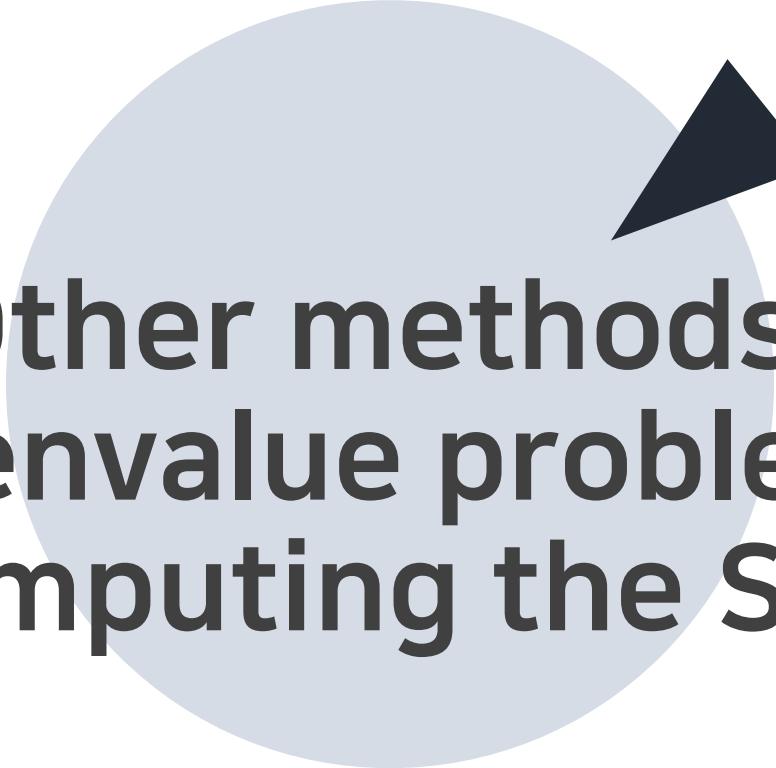
Reverse $A_1 = R_1 Q_1 = \begin{bmatrix} 7.83 & .906 \\ .906 & 3.17 \end{bmatrix}$

Shift $\mu=4$ (A의 right entry)로 QR분해를 계산하면

$$A_0 - \mu I = Q_1 R_1 = \begin{bmatrix} .832 & .555 \\ .555 & -.832 \end{bmatrix} \begin{bmatrix} 3.61 & 1.66 \\ 0 & 1.11 \end{bmatrix}$$

$$A_1 = R_1 Q_1 + \mu I = \begin{bmatrix} 7.92 & .615 \\ .615 & 3.08 \end{bmatrix}$$

=> 비대각성분들은 작아졌고, 대각성분들은 고유값(8,3)에 더 가까워졌다



5. Other methods for eigenvalue problem / Computing the SVD

Other methods for eigenvalue problem

Jacobi algorithm

Jacobi 회전을 반복적으로 적용하여 실대칭 행렬을 대각 행렬로 변환하자!

standard approach는 2×2 submatrices 기반

$$(A) \quad J^T \begin{bmatrix} a & d \\ d & b \end{bmatrix} J = \begin{bmatrix} \neq 0 & 0 \\ 0 & \neq 0 \end{bmatrix}, \quad (B) \quad J = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}, \quad (C) \quad \tan(2\theta) = \frac{2d}{b-a},$$

$$c = \cos(\theta), s = \sin(\theta)$$

Jacobi algorithm은 (B)와 (C)를 기반으로 하는 (A) 변환의 반복되는 형태로
대각 행렬에 가까워질 때까지 반복을 진행한다

Other methods for eigenvalue problem

Jacobi algorithm

A 를 $m \times m$ 대칭행렬로 확장한 경우

$J = 4$ 개 요소를 제외하면 단위행렬인 모습

$\rightarrow A$ 의 왼쪽에 J^T 를 곱하면 A 의 2개 행 변화

$\rightarrow A$ 의 오른쪽에 J 를 곱하면 A 의 2개 열 변화

$\Rightarrow |\cos(\theta)|, |\sin(\theta)| \leq 1$ 이므로 0이 아닌 요소들의 크기가 점점 감소한다

$$J = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

$$A' = J^T AJ$$

$$\begin{aligned} A'_{ij} &= A'_{ji} = (c^2 - s^2)A_{ij} + sc(A_{ii} - A_{jj}) \\ &= (\cos 2\theta)A_{ij} + \frac{1}{2}(\sin 2\theta)(A_{ii} - A_{jj}) \end{aligned}$$

$$A'_{ik} = A'_{ki} = cA_{ik} - sA_{jk} \text{ for } k \neq i, j$$

$$A'_{jk} = A'_{kj} = sA_{ik} = cA_{jk} \text{ for } k \neq j, i$$

$$A'_{ii} = c^2 A_{ii} - 2scA_{ij} + s^2 A_{jj}$$

$$A'_{jj} = s^2 A_{ii} + 2scA_{ij} + s^2 A_{jj}$$

$$A'_{ij} = 0 \rightarrow \tan(2\theta) = \frac{2A_{ij}}{A_{jj} - A_{ii}}$$



Other methods for eigenvalue problem

Jacobi 예제 풀이

$$A = \begin{bmatrix} 1 & \sqrt{2} & 2 \\ \sqrt{2} & 3 & \sqrt{2} \\ 2 & \sqrt{2} & 1 \end{bmatrix}$$

가장 큰 off-diagonal element=2=a₁₃=sinθ

$$\begin{aligned} a_{13} &= \sin\theta \\ a_{31} &= -\sin\theta \\ a_{11} &= \cos\theta \\ a_{33} &= \cos\theta \end{aligned}$$

$$J = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\begin{aligned} \tan(2\theta) &= \frac{2 \times 2}{1-1} = \infty \\ \therefore 2\theta &= \frac{\pi}{2}, \theta = \frac{\pi}{4} \end{aligned}$$

$$\begin{aligned} J^T &= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{-1}{\sqrt{2}} \\ 0 & 1 & 0 \\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}, A = \begin{bmatrix} 1 & \sqrt{2} & 2 \\ \sqrt{2} & 3 & \sqrt{2} \\ 2 & \sqrt{2} & 1 \end{bmatrix}, J = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ \frac{-1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \\ \rightarrow J^T AJ &= \begin{bmatrix} -1 & 0 & 0 \\ 0 & 3 & 2 \\ 0 & 2 & 3 \end{bmatrix} = B \end{aligned}$$

B는 대각 행렬이 아니므로, 2번째 반복을 진행

가장 큰 off-diagonal element=2=a₂₃=sinθ

$$\begin{aligned} a_{23} &= \sin\theta \\ a_{32} &= -\sin\theta \\ a_{22} &= \cos\theta \\ a_{33} &= \cos\theta \end{aligned}$$

$$J = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$\begin{aligned} \tan(2\theta) &= \frac{2 \times 2}{3-3} = \infty \\ \therefore 2\theta &= \frac{\pi}{2}, \theta = \frac{\pi}{4} \end{aligned}$$

$$\begin{aligned} J^T &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}, B = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 3 & 2 \\ 0 & 2 & 3 \end{bmatrix}, J = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \\ \rightarrow J^T BJ &= \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{bmatrix} \end{aligned}$$

$J^T BJ$ 는 대각 행렬이므로, 고유값은 5, 1, -1



Computing the SVD

Connection between the SVD and the eigen decomposition

SVD of A : $U\Sigma V^*$

$$A^*A = V\Sigma^*\Sigma V^*$$

1. Form A^*A

2. Compute the eigenvalue decomposition $A^*A = V\Lambda V^*$

3. Let Σ be the $m \times n$ nonnegative diagonal square root of Λ

4. Solve the system $U\Sigma = AV$ for unitary U

$$AA^* = U\Sigma^2U^*$$

$$A^*A = V\Sigma^2V^*$$

- AA^* 를 통해 U 를 구할 수 있다

- A^*A 를 통해 V 를 구할 수 있다

이러한 형태를 통한 변환은 perturbation에 민감하여 unstable하다!

Computing the SVD

Connection between the SVD and the eigen decomposition

A 는 정방행렬($m = n$)

$$H = \begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix}$$

$A = U\Sigma V^*$ implies $AV = U\Sigma$ and $A^*U = V\Sigma^* = V\Sigma$

$$\begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix} \begin{bmatrix} V & V \\ U & -U \end{bmatrix} = \begin{bmatrix} V & V \\ U & -U \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix}, \quad HP = PD \rightarrow H = PDP^*$$

Figure 14

A 의 singular value는 H 의 eigenvalue의 절댓값이고, A 의 singular vector는 H 의 eigenvector에서 추출할 수 있다
즉, 정방행렬 A 의 SVD는 A 를 바탕으로 H 행렬을 생성한 후 H 의 고유값 분해를 통해 얻을 수 있다

이러한 형태를 통한 변환은 stable하다!

Computing the SVD

Two Phases

SVD 문제는 two-phase 접근이 일반적

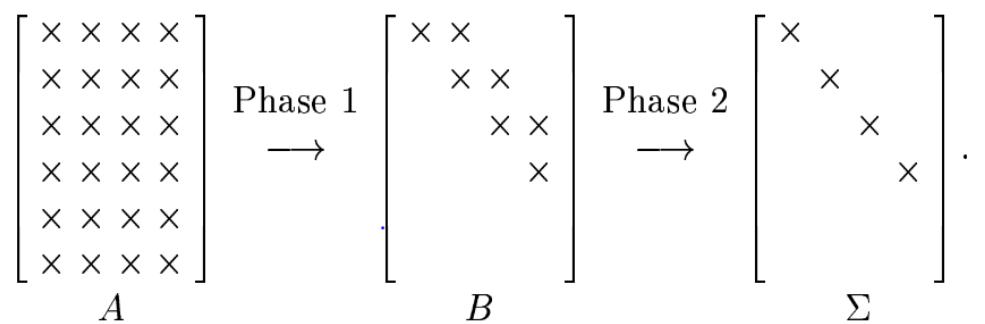


Figure 15

Phase 1- $O(mn^2)$ flops

Phase 2- 이론상으로는 infinite operations 이지만,

실제로는 $O(n)$ iterations이며 각 iteration마다 $O(n)$ flops $\rightarrow O(n^2)$ flops

⇒ Phase 1에 비해 2가 더 깊다

Phase 1: A 행렬을 bidiagonal 형태로 변환한다

$$A = UBV^*$$

Phase 2: bidiagonal 행렬을 diagonal matrix 형태로 변환한다

$$B = U' \Sigma (V')^*$$

Phase 1과 Phase 2를 종합하면

$$A = (UU')\Sigma(VV')^*$$



Computing the SVD

Golub-Kahan Bidiagonalization

(Phase 1) 좌우에 Householder reflector를 번갈아 적용하여 0을 만들자! - 고유값을 위한 Householder reduction 과정과 유사한 형태
bidiagonal matrix \leftrightarrow upper Hessenberg matrix

좌- 대각 원소 아래를 0으로 바꾼다 (1~m, 2~m, … 행 변화)

우- first superdiagonal의 오른쪽을 0으로 바꾼다 (2~n, 3~n, … 열 변화)

$$\begin{aligned} A &= \begin{bmatrix} x & x & x & x \\ x & x & x & x \end{bmatrix}. \quad U_1^* A = \begin{bmatrix} x & x & x & x \\ 0 & x & x & x \end{bmatrix} \rightarrow U_1^* A V_1 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & x \end{bmatrix} \\ &\quad \rightarrow U_2^* U_1^* A V_1 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{bmatrix} \rightarrow U_2^* U_1^* A V_1 V_2 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{bmatrix} \\ &\quad \rightarrow U_3^* U_2^* U_1^* A V_1 V_2 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & x \end{bmatrix} \rightarrow U_4^* U_3^* U_2^* U_1^* A V_1 V_2 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 \end{bmatrix} = B. \end{aligned}$$

Figure 16

Computing the SVD

Golub-Kahan Bidiagonalization

진행 과정이 A 와 A^* 에 Householder QR 분해를 한 형태와 유사하므로 total operation count는 QR 분해의 2배
 $\sim 4mn^2 - 4/3 n^3$ flops

Golub-Kahan bidiagonalization을 바탕으로 Phase 1의 flop을 줄이는 방향으로
LHC bidiagonalization, Three-step bidiagonalization 순서로 발전

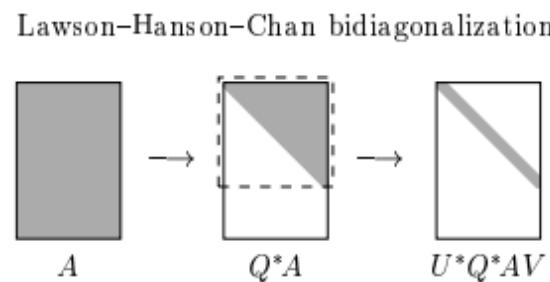


Figure 17

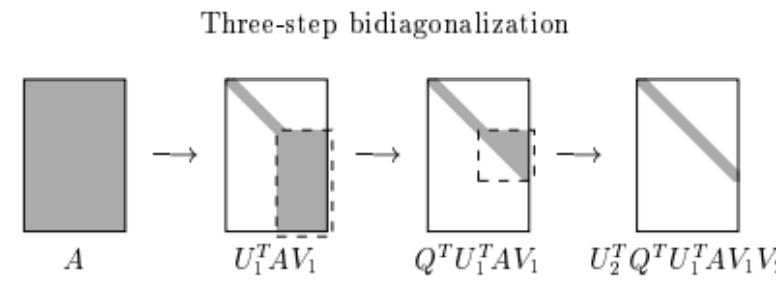


Figure 18

END

References

[1] Darve, E., & Wootters, M. (2021). Numerical Linear Algebra with Julia. SIAM.

[2] Trefethen, L. N., & Bau , D. (1997). Numerical linear algebra (1st ed.). SIAM.

Figure

[1][2] Fasshauer, G. (2006). 8 eigenvalue problems - IIT. 477/577 Handouts and Worksheets. http://math.iit.edu/~fass/477577_Chapter_8.pdf

[3] Trefethen, L. N., & Bau , D. (1997). Numerical linear algebra (1st ed.) (p.194). SIAM.

[4] Trefethen, L. N., & Bau , D. (1997). Numerical linear algebra (1st ed.) (p.194). SIAM.

[5][6] Wikimedia Foundation. (2023, March 29). Rayleigh quotient iteration. Wikipedia. https://en.wikipedia.org/wiki/Rayleigh_quotient_iteration

[7] Trefethen, L. N., & Bau , D. (1997). Numerical linear algebra (1st ed.) (p.208). SIAM.

[8] Darve, E., & Wootters, M. (2021). Numerical Linear Algebra with Julia (p.160). SIAM.

[9] Trefethen, L. N., & Bau , D. (1997). Numerical linear algebra (1st ed.) (p.197). SIAM.

[10] Trefethen, L. N., & Bau , D. (1997). Numerical linear algebra (1st ed.) (p.198). SIAM.

[11] Darve, E., & Wootters, M. (2021). Numerical Linear Algebra with Julia (p.170). SIAM.

[12] Darve, E., & Wootters, M. (2021). Numerical Linear Algebra with Julia (p.179). SIAM.

[13] Darve, E., & Wootters, M. (2021). Numerical Linear Algebra with Julia (p.181). SIAM.

[14] Trefethen, L. N., & Bau , D. (1997). Numerical linear algebra (1st ed.) (p.235). SIAM.

[15] Trefethen, L. N., & Bau , D. (1997). Numerical linear algebra (1st ed.) (p.236). SIAM.

[16] Trefethen, L. N., & Bau , D. (1997). Numerical linear algebra (1st ed.) (p.237). SIAM.

[17] [18] Trefethen, L. N., & Bau , D. (1997). Numerical linear algebra (1st ed.) (p.238). SIAM.

