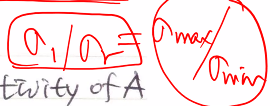
**Chap 7. Computation with Matrices**

Numerical computation임.

**7.1 Introduction**

Ax=b와 Ax = 람다x가 큰 이슈임.

3. Matrix의 condition number는 

Eigenvalue로 계산.



A와 A^-1이 수치적으로 얼마나 안정적인지 계산.

이번 챕터에서는 모든 matrix가 square라고 가정.

1. Technique for Solving Ax = b

Elimination이 perfect한 algorithm임. 특정 성질을 가진 문제들 제외 – 근데 대부분 모든 문제들이 가지고 있음.

Sparse matrix는 Ax = b를 푸는 iterative method가 더 나을 수가 있음.

목적은 소거법보다 더 빨리 해를 찾는 것. 근데 많은 경우 소거법이 더 안전하고 빠름. 특히 0이 많을 때.

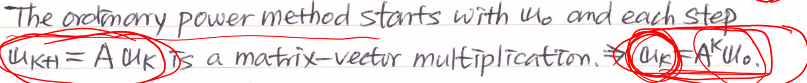
방법들은 서로 경쟁 중.

2. Techniques for solving Ax = 람다x

최근까지 아무도 eigenvalue 문제를 푸는 방법을 몰랐음. 열 몇 개의 알고리즘이 소개됨.

두, 세개 정도가 살아남음. QR 알고리즘이 대표., power method, symmetric matrix를 tridiagonal로 만들기 위해 전처리하는 것. 얘는 direct method지만 더 간단한 행렬을 만듦.

**7.3 Computation of Eigenvalues**



일반적으로는 요렇게 계산함.

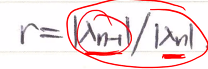
이 때 A로 곱하는 건 쉬워야 하고 큰 매트릭스 A는 sparse 해야 함. 그 경우에 power method 사용하기 쉬움.

A가 eigenvalue의 full set을 가지고 있고 x1-xn이 연관된 eigenvector일 때 u\_k는 요렇게 나옴.

람다1 <= 람다2 <= … < 람다 n이라고 할 때, c\_n != 0이면 cn\*xn가 u\_k를 지배함.

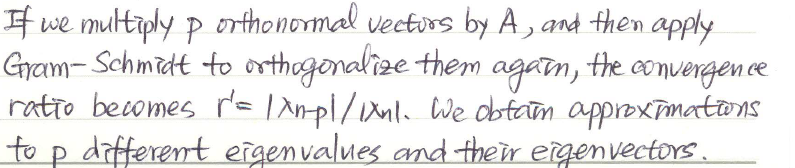


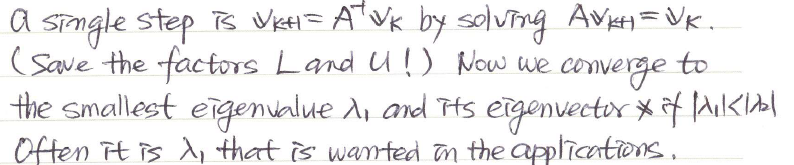
U는 eigenvalue 가 가장 큰 eigenvector(cn\*xn)로 수렴하게 됨.

 이게 1이면 굉장히 늦게 수렴하고, 작을 수록 빨리 수렴함.

R = 1이면 수렴하지 않음.

이 한계에 가는 방법은 여러 개 있음.

1. block power method는 한번에 여러 개 vector와 함. 

2. inverse power method는 A 대신 A^-1에 적용함. 

가장 작은 람다1을 찾는 것.

A^-1 \* x\_i = (1 / 람다\_i) \* x\_i가 됨.

LU \* v\_k+1 = v\_k

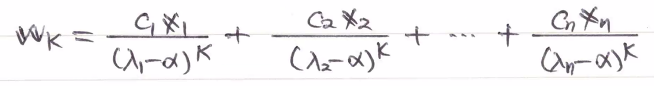
LC\_k+1 = v\_k

Uv\_k+1 = c\_k+1

이렇게 풂. 수치적으로 안정됨.

3. 근데 shifted inverse power 방법이 가장 좋음.

A를 로 대체함. 각각의 eigenvalue가 알파만큼 이동하고, 인버스의 convergence factor는 가 됨. 알파가 람다1의 좋은 예측이면, r’’은 엄청 빠르게 작아지고 수렴함. 각각은 를 풂.



알파가 람다1과 가까우면 첫 번째 식이 지배함. Any eigenvalue에 대한 좋은 approximation이 다른 알고리즘(QR)에 의해 계산되면, 이건 eigenvector를 찾기 위한 좋은 방법임.

만약 람다1이 미리 예상되지 않았으면, 이 방법은 알파를 정해야 함. 각 단계에서 알파를 다르게 할 수 있음. 만약 A가 symmetric이면, 좋은 방법은 Rayleigh quotient.



여기서 R(x)는 아이겐벡터 x\_1에서 최솟값을 가짐. 에러는 eigenvector가 가지고 있는 오차의 제곱이 됨.





**Tridiagonal and Hessenberg Forms**

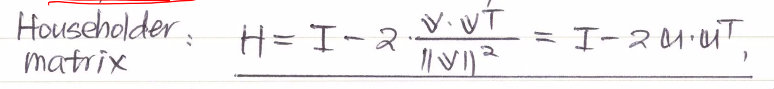


Power method는 큰 sparse matrix한테만 의미 있음. 그래서 0을 만드는 간단한 방법을 찾는 게 목표.

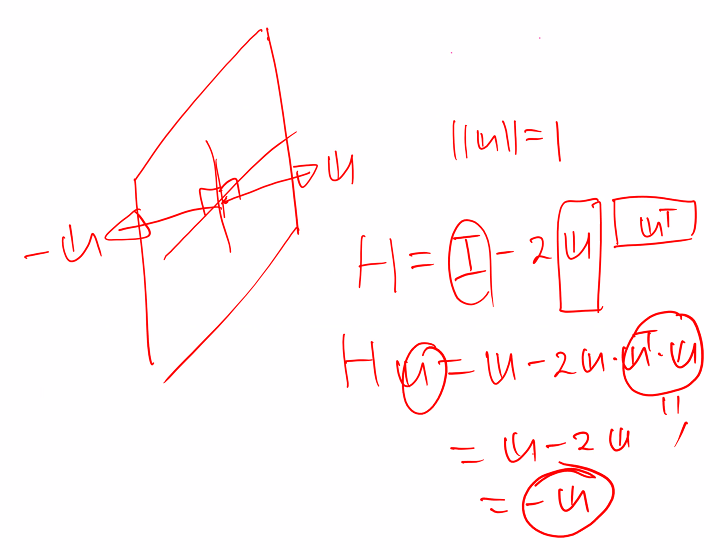
Similarity Transform 은 A보다 0이 많음. 그러면 이제 power method로 돌아갈 이유가 없음. QR 알고리즘이 훨씬 강력함. (shifted inverse power method가 마지막에 등장)

첫번째 스텝은 Q(an orthogonal matrix)를 이용해서 빠르게 0을 많이 만드는 것. A가 symmetric이면, 도 symmetric임.

Hessenberg form은 하나의 0이 아닌 nonzero 대각선을 대각선 밑에 허용함. symmetric이면 단 3개의 0이 아닌 대각이 있게 됨. 

Householder matrix는 하나의 벡터 v에 의해 결정되는 reflection matrix임. 

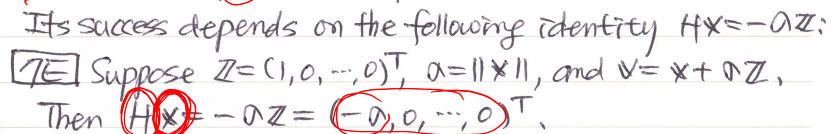


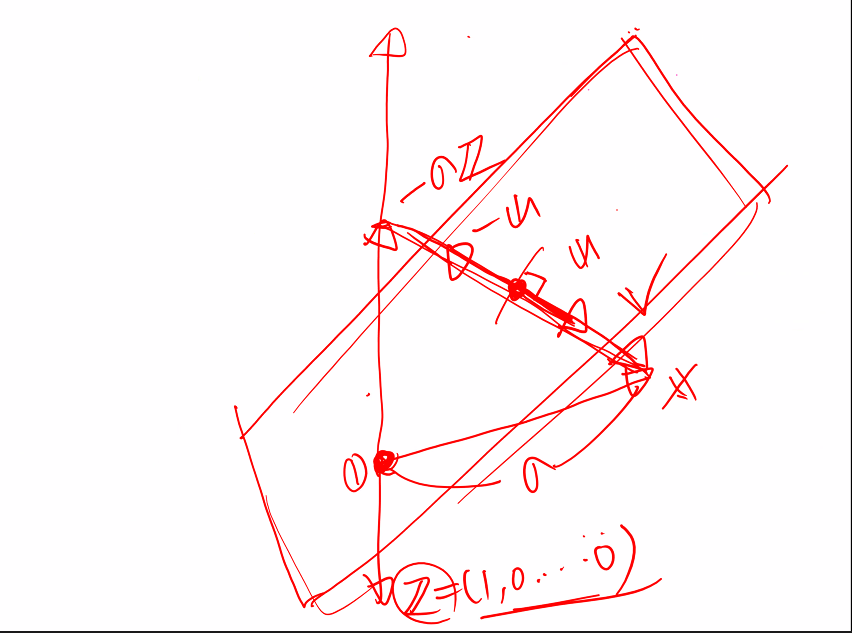


Hu = -u가 되므로 reflection.

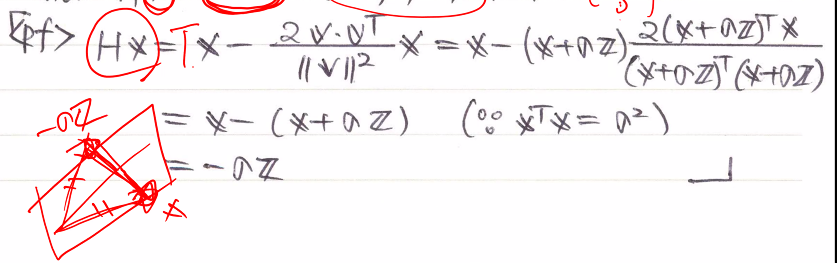
H는 symmetric이고 orthogonal임. H = H^T = H^-1.

Householder의 계획은 매트릭스를 이용해서 0을 만드는 것.

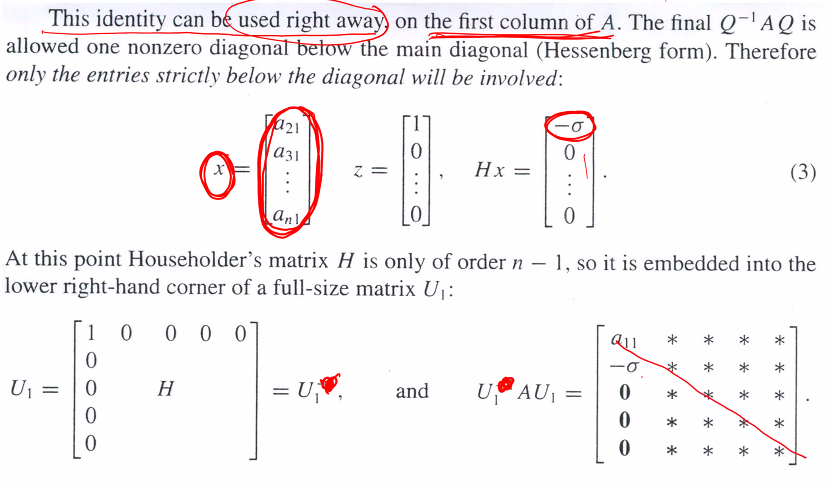




Z, v, az 등으로 만들어지는 평면.

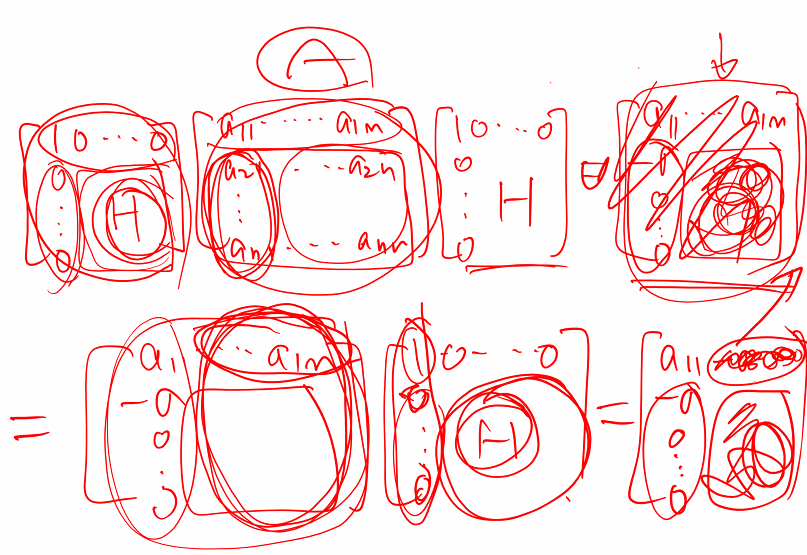


이렇게 x에 H를 곱하면 x랑 -az랑 reflection 됨.

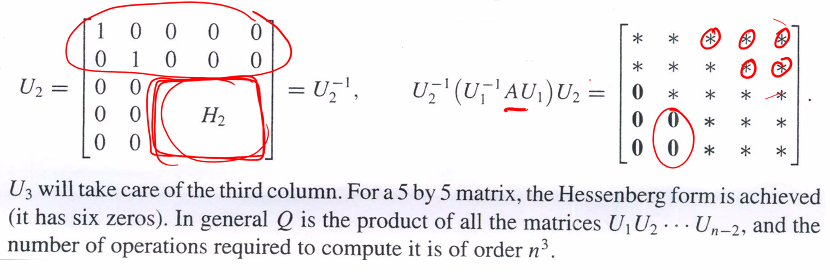


첫번째 칼럼만 됨. 헤센버그로 바꾸려면 Lower left가 전부 0이 되어야 함.

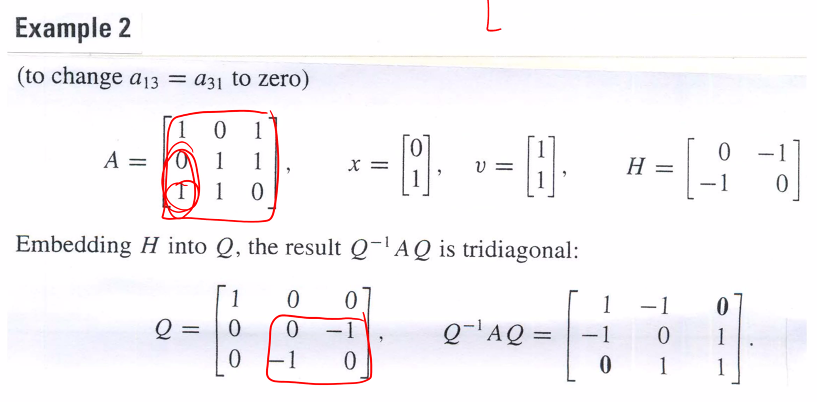
U2, U3, U4… 등을 곱해야 함.

H만 곱하면 eigenvalue가 보존이 안 돼서 H^-1을 또 곱해줌. 근데 이건 자기자신..

symmetric이면 오른쪽 위도 -시그마 0000 이고 아니면 뭐가 나올지 모름.

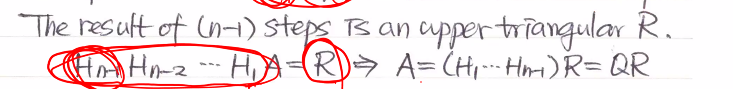


이런 식으로 U2 곱하면 두 번째 column이 0이 됨. 마찬가지로 H3 까지 곱하면 tridiagonal 혹은 헤센버그 행렬이 나옴.

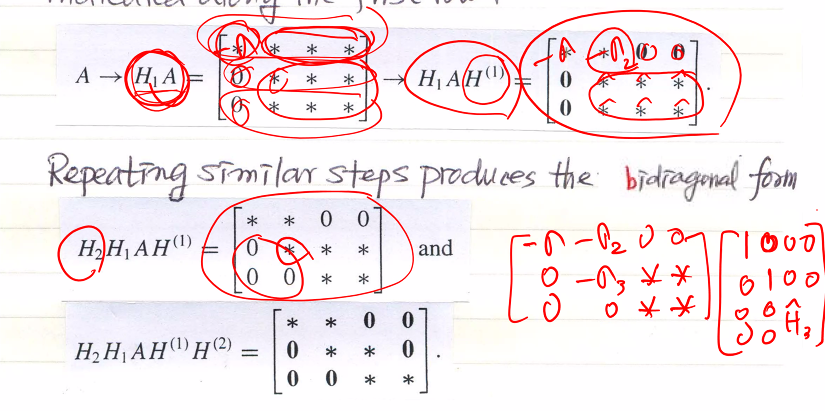


H는 또 Gram-Schmidt factorization A = QR에 쓰일 수 있음.

내용은 pdf 참조.

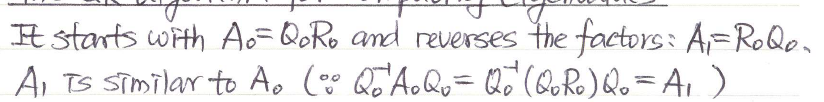


그리고 Singular Value Decomposition 에 쓸 수 있음.

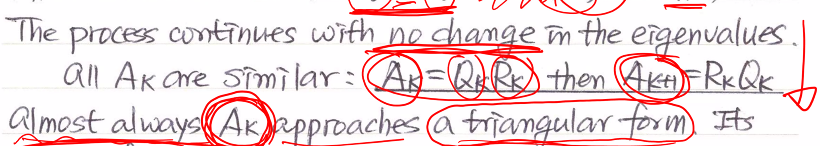


SVD는 아니고 SVD랑 비슷하게 나옴. Bidiagonal.

**The QR Algorithm for Computing Eigenvalues**

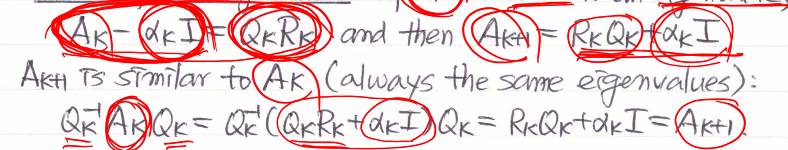


A0 = Q0R0, A1 = R0Q0 하면 A1은 A0과 similar.



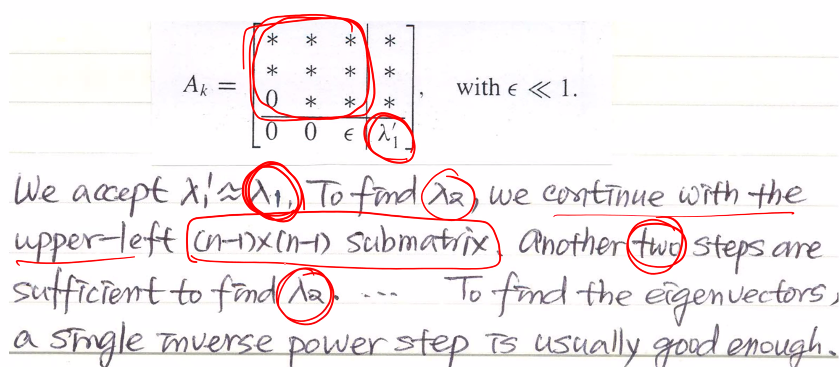
Shifting 하면 더 빨리 수렴함.

1. The Shifted Algorithm:

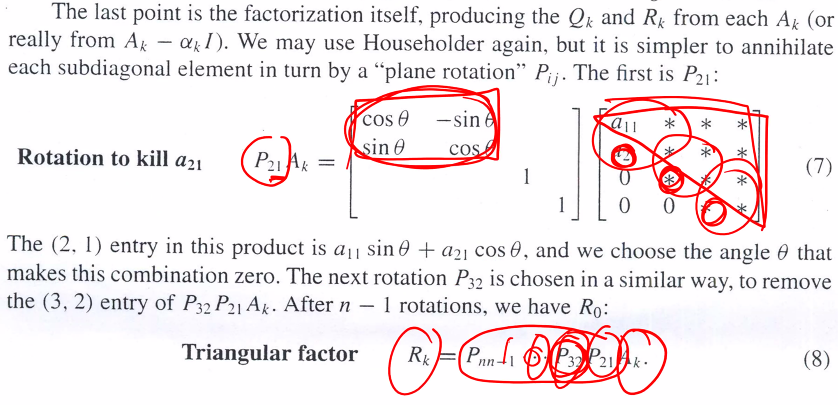


이렇게 shift 해도 similar함.

엄청 빠르게 근사함. 세~네번만 돌리면 람다1에 근사.



2. A가 tridiagonal이나 Hessenberg면 QR이 매우 빠름. Gram-Schmidt가 O(N^3)인데 Hessenberg면 O(N^2), tridiagonal이면 O(N). 다행히도, 각각의 새 A\_k 또한 Hessenberg or tridiagonal form임.



A에서 Rotation을 계속 곱해서 R을 만들 수 있음.

Rotation은 Orthogonal Transform.

**7.4. Iterative Methods for Ax = b**

A = S - T라고 하면, Ax = b는 Sx = Tx + b와 같음.

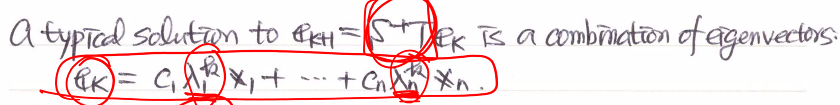
Iteration: 

1. 이 새로운 벡터는 계산하기 쉬워야 함, 그러려면 S는 간단하고 invertible 해야함. – diagonal or triangular.
2. Sequence x\_k는 true solution으로 converge 해야 함. 

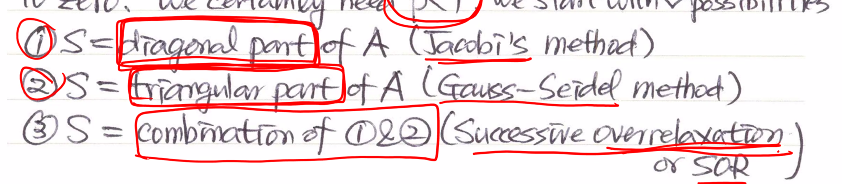
7F – . 요 계산한 행렬의 모든 eigenvalue가 1보다 작아야 함!



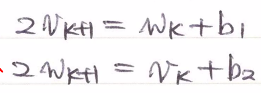
S는 또한 preconditioner라고도 불림, 그리고 그건 numerical analysis에서 중요.



가장 큰 eigenvalue가 결국은 지배하고, spectral radius가 e\_k가 수렴하는걸 지배함.

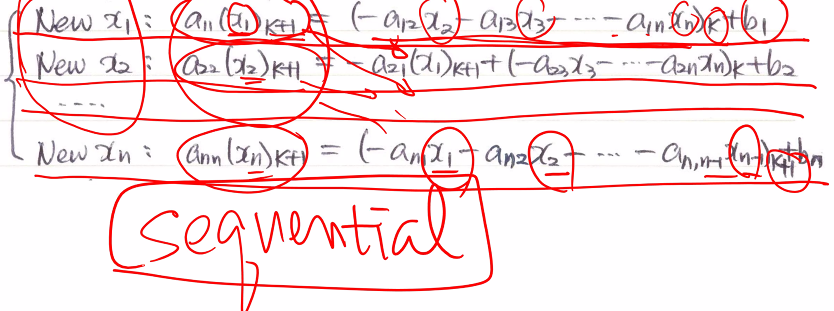


Ex1: Jacobi 사용. S를 A의 대각선 원소만 남기고 T는 sign을 바꿈. S^-1T는 eigenvalue가 +- 1/2을 가짐. 그래서 error가 매 step마다 1/2씩 줄어듦.



요건 병렬로 수행됨.

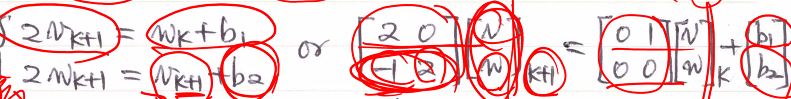
Gauss-Seidel method: 새 x\_k+1의 각각의 component를 계산되자마자 바로 씀. 그러면 jacobi보다 공간을 반틈 먹음.



얘는 순차적. 하지만 parallel 하지 않음.

Ex2: Gauss-Seidel

S를 left matrix만 택하고 T는 윗부분에 sign을 바꿈.



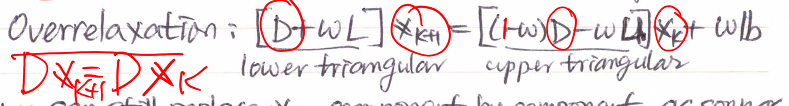
V\_k+1이 계산되자 마자 다음 줄에서 적용됨.

Eigenvalue는 1/4, 0임. 에러는 각 step마다 4로 나눠짐. 그래서 한 번의 Gauss-Seidel step이 두 번의 Jacobi steps와 같다. 하지만 Jacobi는 수렴하는데 Gauss-Seidel이 수렴하지 않는 경우 (혹은 그 반대)도 있음.

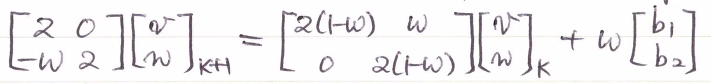
SOR은 둘보다 훨씬 더 빠름! 오메가가 1이면 Gauss-Seidel이 되고, w>1이면 SOR. Successive Overrelaxation. 보통 최적의 w는 2를 넘지 않고, 1.9가 보통임.

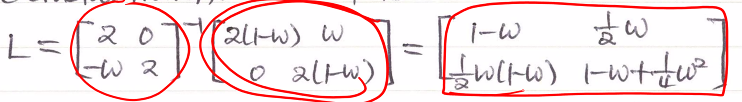
D, L, U를 대각선, 아래, 위라고 하자 (A=LDU 분해와 다름!). 그러면 A = L + D + U.

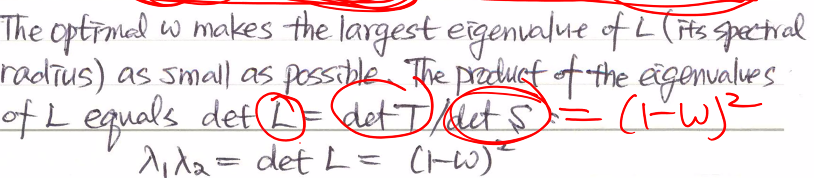
Jacobi는 S = D, T = -L-U, Gauss-Seidel은 S = D+L, T = -U.

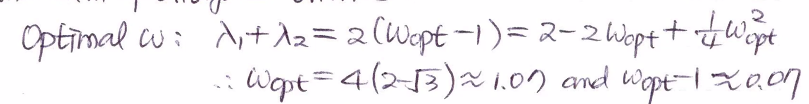


EX3: SOR 사용. 같은 행렬 A에 대해서, SOR은



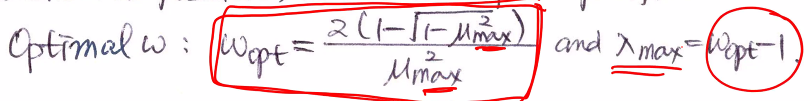
L = S^-1T에서 

그래서 det는 . 각각 삼각 행렬이니 대각선 원소들의 곱이 eigenvalue이고 eigenvalue의 곱과 det가 같음.

이 예제에서는 

(1/4)^2 이 0.07이랑 비슷하기 때문에 기존의 Gauss-Seidel을 두 번 한 게 한 번과 비슷.



크기가 더 큰 행렬에서는

최적의 w가 이렇게 나옴.