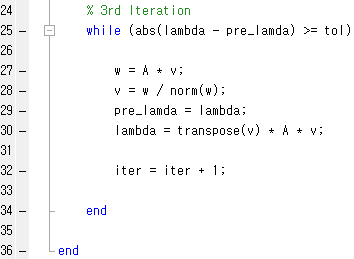
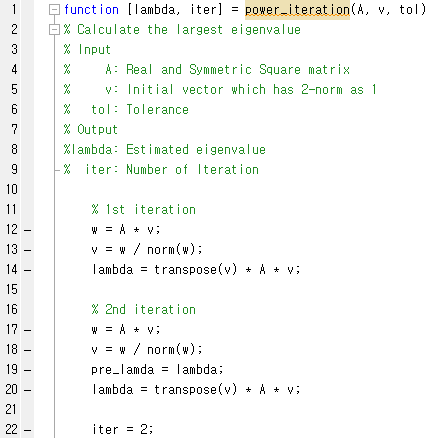
**MATLAB HW#8**

20150651 장강욱

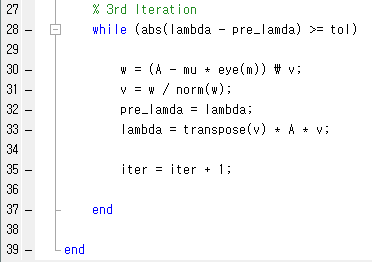
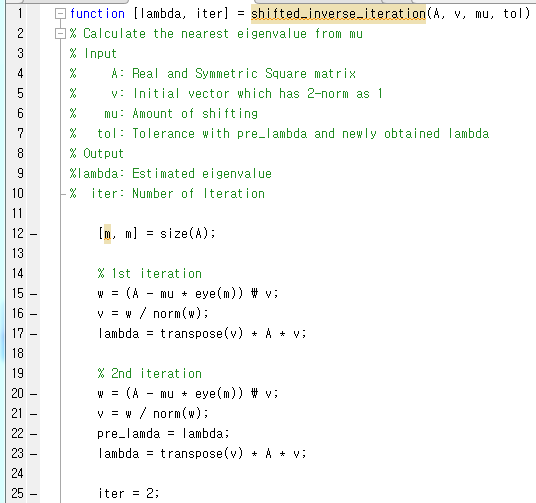
Dept: EE

1. (a)

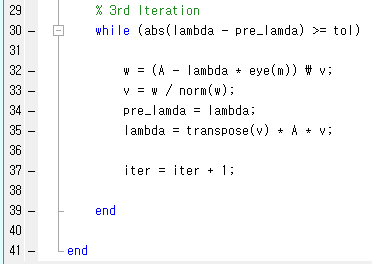
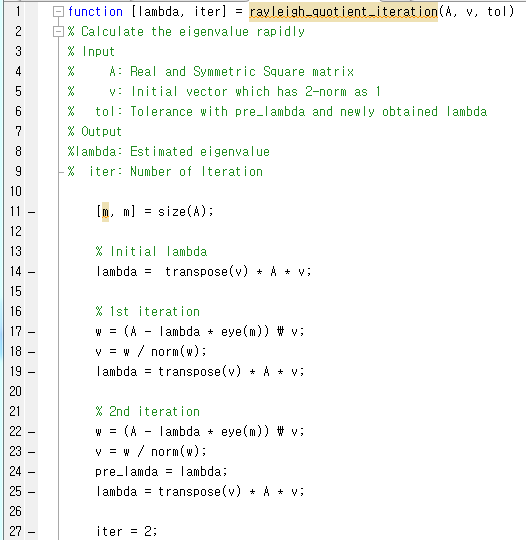
아래는 Power iteration에 대한 코드이다. 이전에 얻은 고윳값 결과와 비교하기 위해, 첫 두 번의 반복은 직접 코드를 적은 후, 변수 pre\_lambda에 첫 근사를 대입하였다. 메소드에 입력되는 행렬 A와 벡터 v는 같은 열 크기를 가지며, 행렬 A는 Real & Hermitian 정사각행렬이라 가정하였다. 메소드의 입력과 출력은 주석을 확인하자.



아래는 Shifted inverse iteration에 대한 코드이다. 이전에 얻은 고윳값 결과와 비교하기 위해, 첫 두 번의 반복은 직접 코드를 적은 후, 변수 pre\_lambda에 첫 근사를 대입하였다. 메소드에 입력되는 행렬 A와 벡터 v는 같은 열 크기를 가지며, 행렬 A는 Real & Hermitian 정사각행렬이라 가정하였다. 메소드의 입력과 출력은 주석을 확인하자.



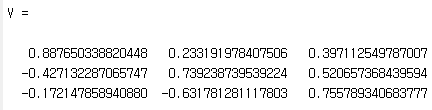
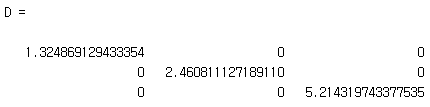
아래는 Rayleigh Quotient iteration에 대한 코드이다. 이전에 얻은 고윳값 결과와 비교하기 위해, 첫 두 번의 반복은 직접 코드를 적은 후, 변수 pre\_lambda에 첫 근사를 대입하였다. 메소드에 입력되는 행렬 A와 벡터 v는 같은 열 크기를 가지며, 행렬 A는 Real & Hermitian 정사각행렬이라 가정하였다. 메소드의 입력과 출력은 주석을 확인하자.



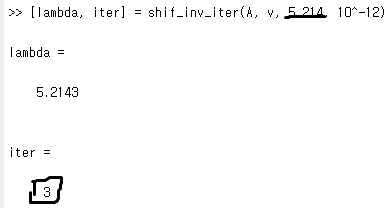
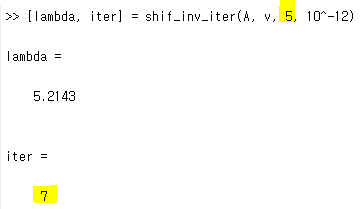
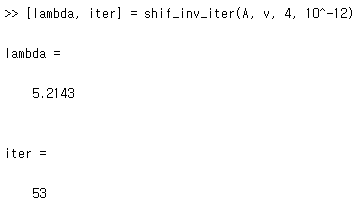
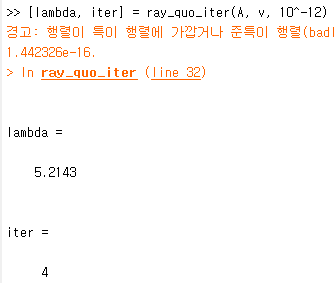
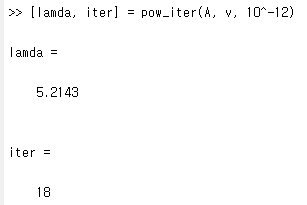
(b)

eig 메소드를 써서, 행렬 A의 고유값과 고유벡터를 구하였다. 출력의 대각행렬의 고윳값 순서와 고유벡터 행렬의 열벡터 순서가 서로 대응되는 고윳값과 고유벡터이다.

EMB00001a9c4c6c

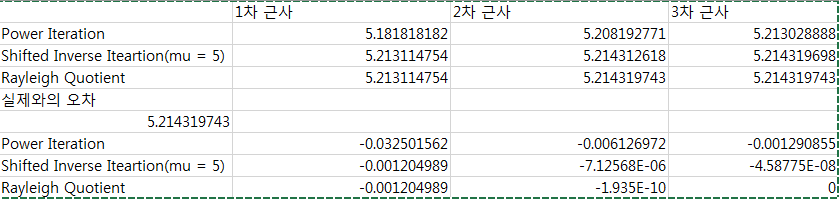


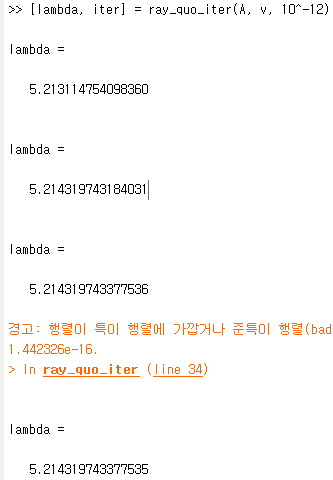
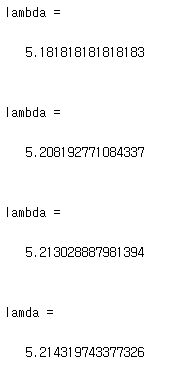
©

Power iteration, Shifted inverse iteration, Rayleigh Quotient Iteration 알고리즘을 차이 10^-12를 가질 때까지 반복하였다. Shifted inverse iteration의 경우, 초기 DRW00001a9c4c6f를 4로 두었다.  


반복 횟수(수렴 속도) 면에서 정리하자면, Rayleigh Quotient Iteration이 항상 Power Iteration보다 빠르다. Shifted Inverse Iteration은 시작하는 DRW00001a9c4c76의 값에 따라 반복횟수가 많이 차이 났다. 초기값을 고려하지 않고 항상 빠른 속도를 원한다면 Rayleigh Quotient Iteraiton이 제일 좋고, 구하려는 고윳값의 범위가 충분히 한정되어 있다면 Shifted Inverse Iteration도 좋은 방법이다.

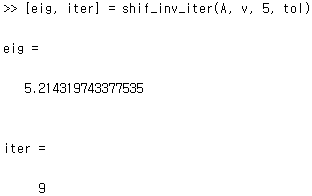
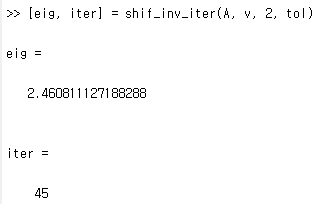
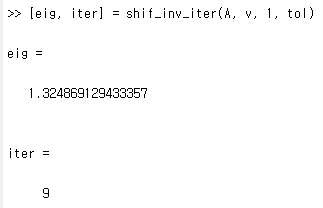
그 다음으로는 코드를 조금 수정하여, 첫 3번의 고윳값을 출력하도록 하였다. 아래는 그것을 정리한 표이다. 쉽게 알 수 있듯이, 실제로 수렴하는 속도는 Rayleigh Quotient, Shifted Inverse, Power Iteration 순서였다.



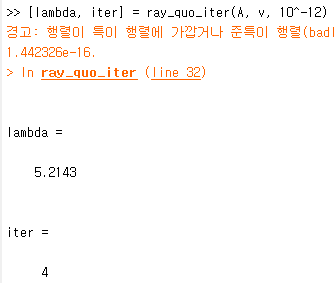
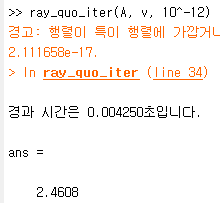
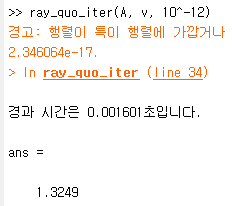


(d)

(b)에서 고윳값을 확인할 수 있었다. 이것을 바탕으로 Shifted Inverse Iteration을 통해 근사된 고윳값 3개를 모두 구할 수 있다. (b)에서 초기 DRW00001a9c4c7c값이 얼마냐에 따라 반복횟수가 상이함을 확인할 수 있다.



Rayleigh Quotient 방법에 대해서도 초기 벡터를 다르게 하면서 모든 고윳값들을 구할 수 있다.



(c)에서 확인할 수 있듯이, Convergence Speed와 실제 고윳값과의 오차는 Rayleigh Quotient 방법이 제일 빠르게 수렴함을 확인할 수 있었다. 그러나 실행시간 측면에서는 아래처럼 Power Iteration 방법이 제일 빠르게 수렴하였다.

EMB00001a9c4c83EMB00001a9c4c84