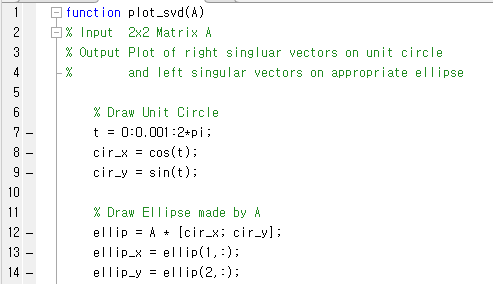
MAS364 Matlab HW #2

20150651 장강욱

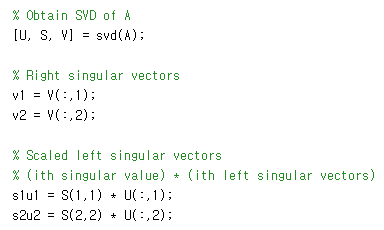
1.

임의의 2x2 행렬 A에 대해 특이벡터(singular vector)들을 도식화하는 함수를 구현한다. 이를 함수 <plot\_svd>로 이름 짓는다.

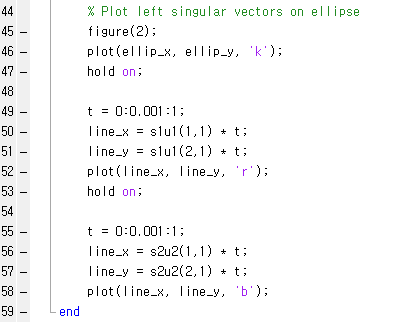
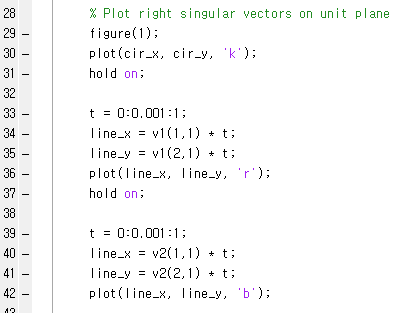
처음에는 단위원과 이것이 행렬 A에 대해 사상(mapping)되는 타원을 그린다. 이를 위해 매개변수에 t를 충분히 작은 간격으로 잡아, 계산된 각 점의 x, y좌표를 행렬(cir\_x, cir\_y, ellip\_x, ellip\_y)로 저장한다.



입력으로 받은 2x2 행렬 A를 SVD하고, 열벡터로 표현되는 특이 벡터들을 뽑아낸다. 좌특이벡터는 타원에 놓아야 하므로, 순서에 대응되는 특이값들을 각각 곱한다.

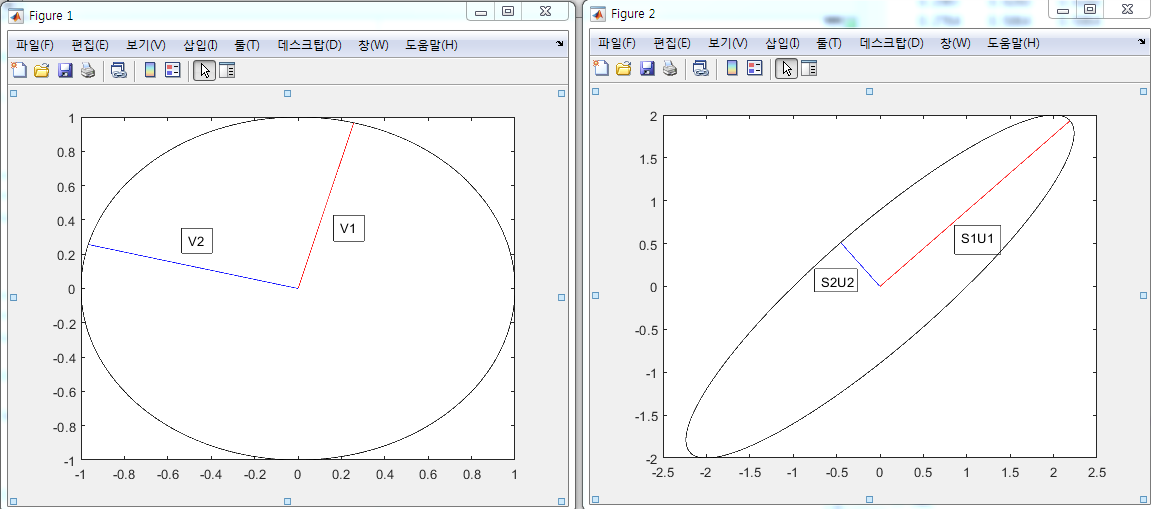


마지막으로, 다시 한 번 매개변수 t를 이용해 각 특이벡터가 나타내는 직선을 단위원과 타원 위에 그린다. 더 나은 도식화를 위해, 원과 두 특이벡터는 다른 색으로 표시한다.

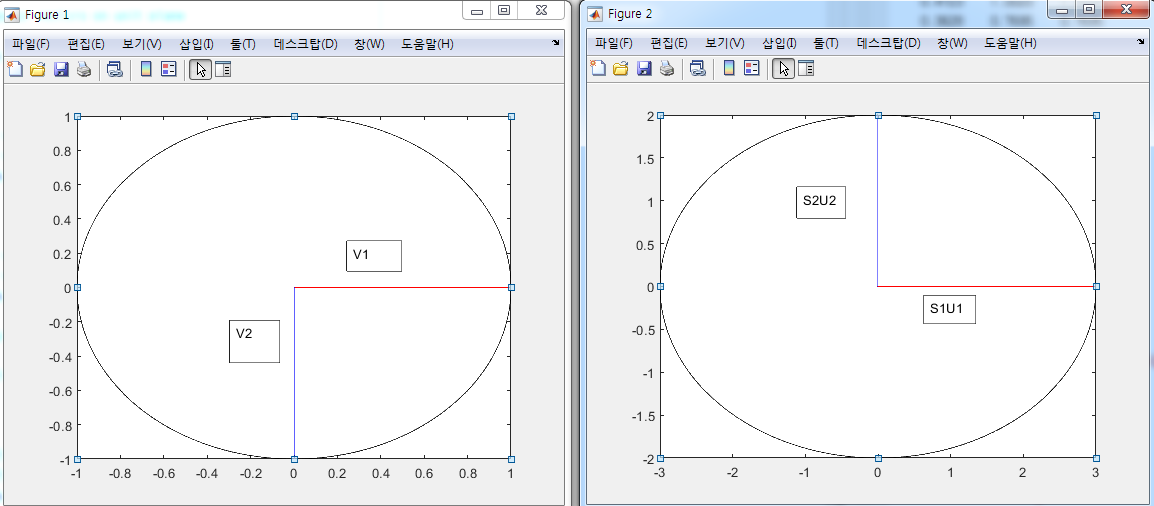


이제 책의 행렬 (3.7)과 연습문제 4.1에서 등장한 모든 2\*2 행렬에 대해 적용해 보자.

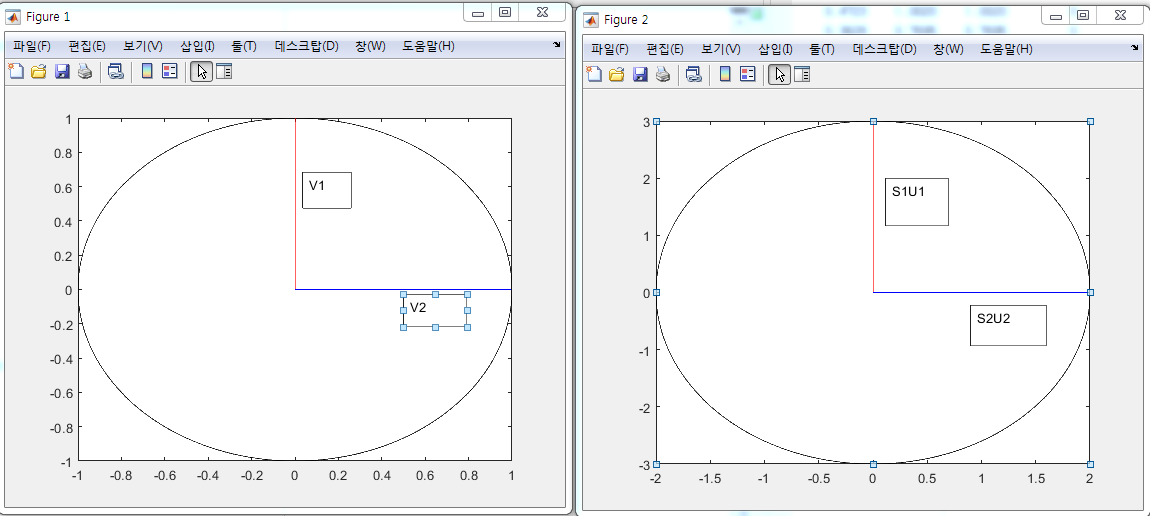
1) DRW00000ef84b91



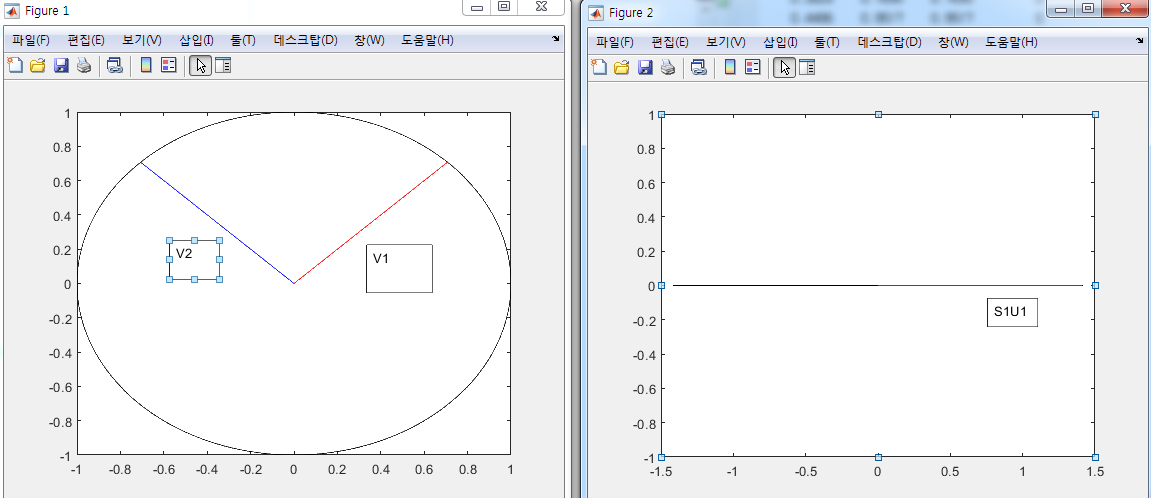
2) DRW00000ef84b94



3) DRW00000ef84b97

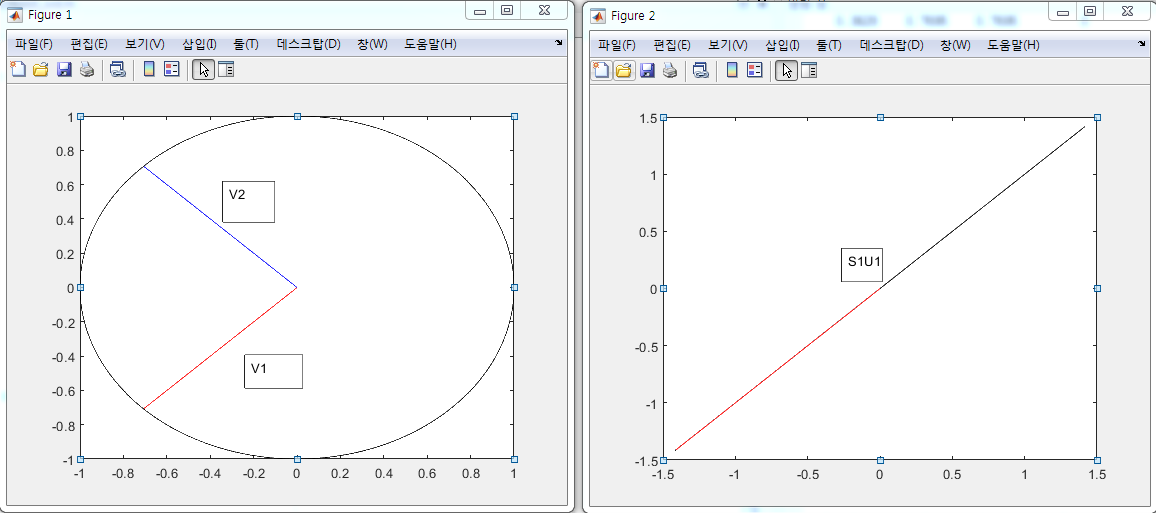


4) DRW00000ef84b9a



이 행렬은 특이값이 1개 밖에 존재하지 않아, 타원이 아닌 장축 DRW00000ef84b9d, 단축이 0인 직선으로 사상되었다.

5) DRW00000ef84b9f

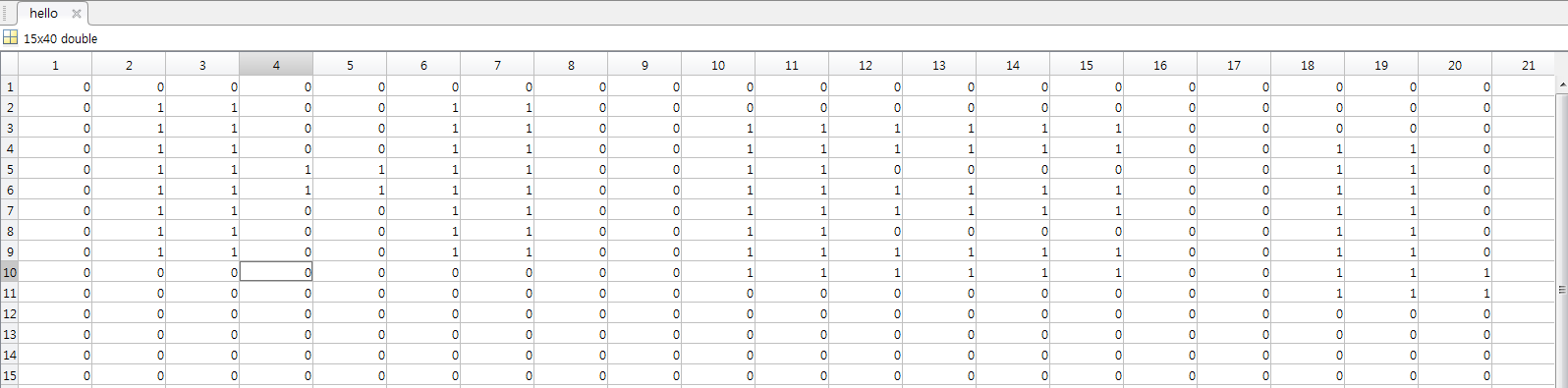


4)와 마찬가지로, 주어진 행렬의 특이값이 1개만 존재한다. 따라서, 단위원에 대한 사상이 타원이 아닌 직선으로 나타났다.

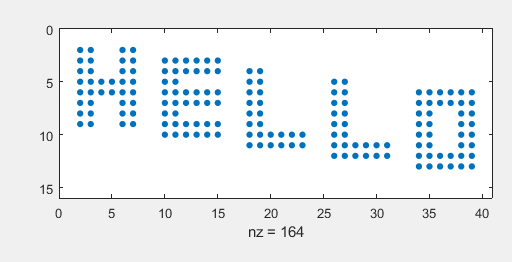
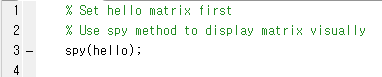
2. (a)

(모든 코드에서 변수 hello는 문제에서의 행렬 A를 의미한다.)

15 x 40 의 행렬을 작성하기 위해 매트랩 상단 도구 모음에서 <새 변수> 기능을 이용했다. 문제에서 주어진 검은색 부분에 1을 채우면, 나머지는 0이 자동으로 채워진다. 이것을 이용해 쉽게 행렬을 설정할 수 있다.

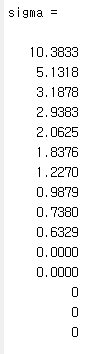
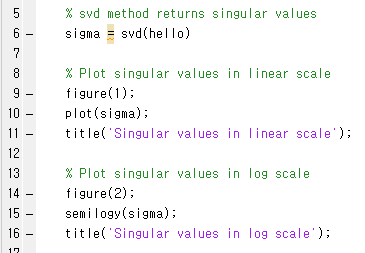


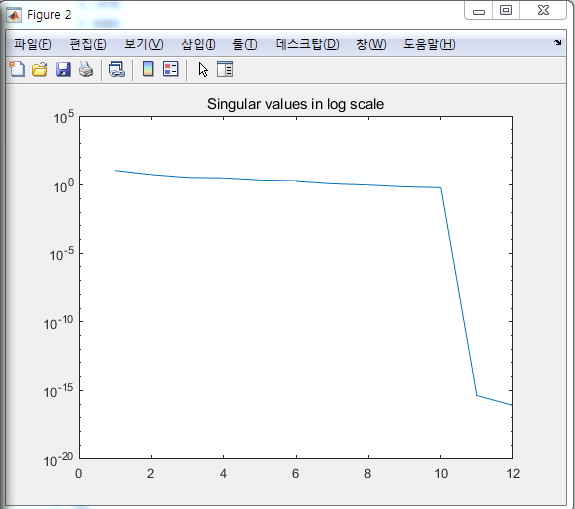
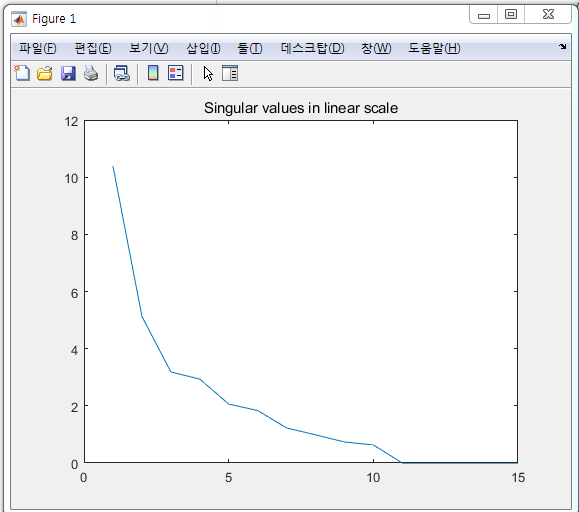
행렬을 다 작성한 뒤에는, <spy> 메소드로 주어진 문자 모양 ‘HELLO’를 출력할 수 있다. 다음은 코드와 그 결과이다.



2. (b)

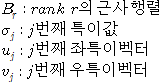
(a)에서 설정한 행렬 A를 <svd> 메소드로 SVD를 진행한다. 이 메소드의 출력은 특이값(Singular Value)들이다. 따라서, 변수 sigma에 <svd> 메소드의 반환을 받아 <plot>과 <semilogy> 메소드로 도식화한다. 다음은 코드와 그 결과이다.



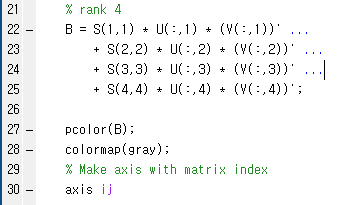


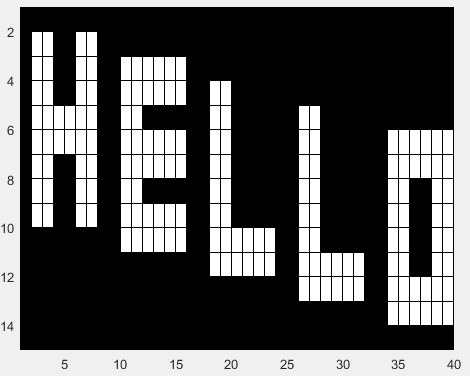
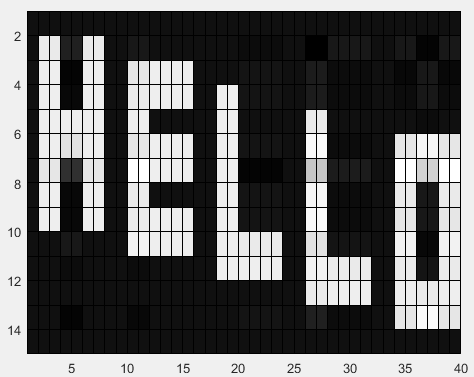
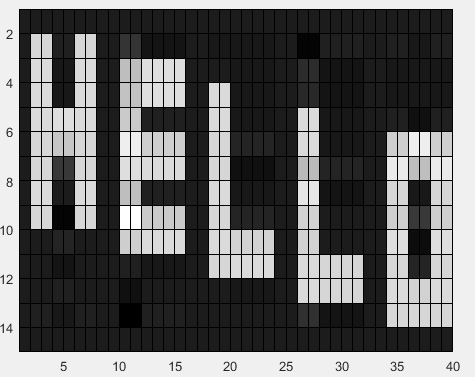
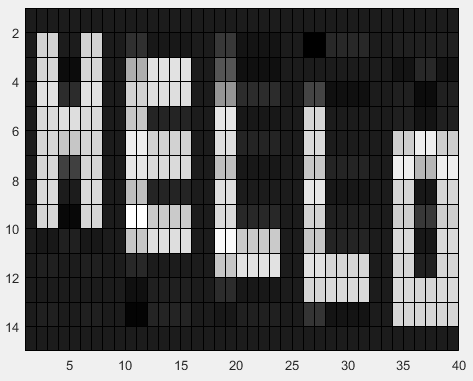
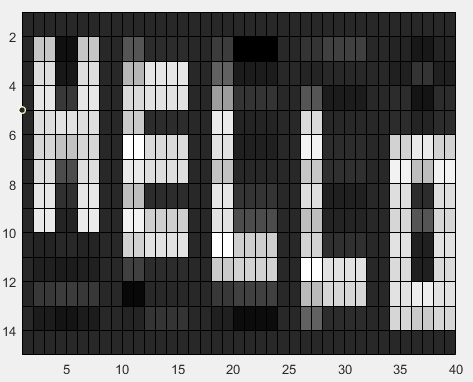
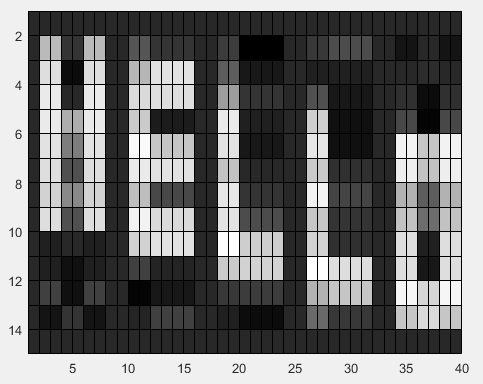
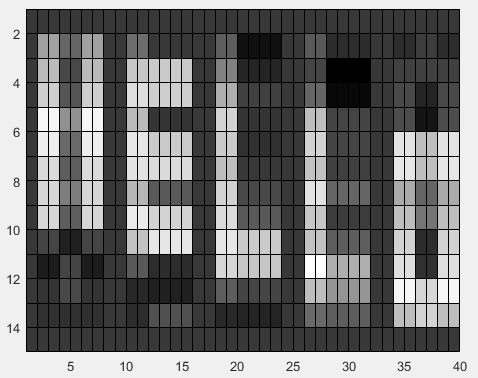
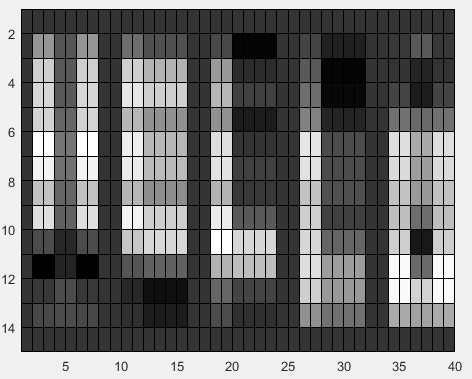
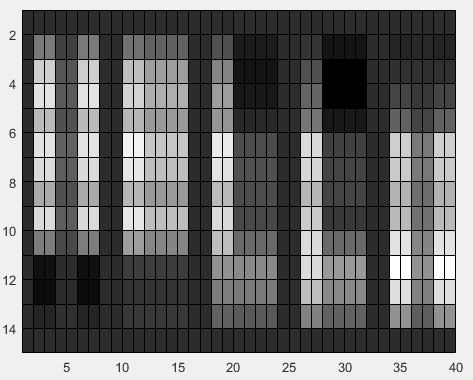
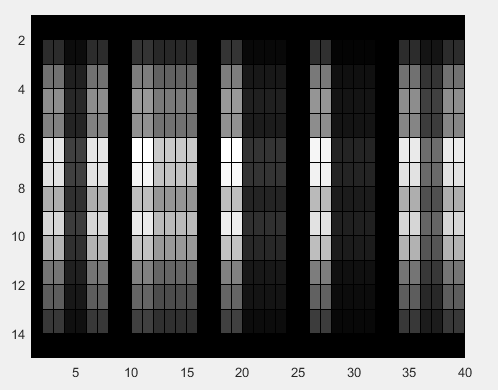
행렬 A의 SVD에서 대각행렬에 존재하는 0이 아닌 특이값의 개수가 곧 A의 rank이다. 변수 sigma에서 0이 아닌 특이값의 개수는 10개이다. 이것은 위의 선형/로그 스케일 그래프에서도 확인할 수 있다. 11번째 특이값부터는 단위가 10^-15 미만으로 10번째 특이값에 비해 매우 극히 작아진다. 따라서 행렬 A의 rank는 10임을 알 수 있다.

2. (c)

DRW00000ef84ba9(b)에서 행렬 A가 rank 10이므로, 이를 이용하여 1부터 10까지 근사 행렬 B를 구해본다. 이 때, 주의할 점은 특이값이 내림차순으로 정렬되어 있으므로, rank가 커질수록 후순위의 특이값과 그에 상응하는 위치의 왼쪽과 오른쪽 특이벡터를 참조하게 된다. 아래는 코드에 사용한 연산식이다.

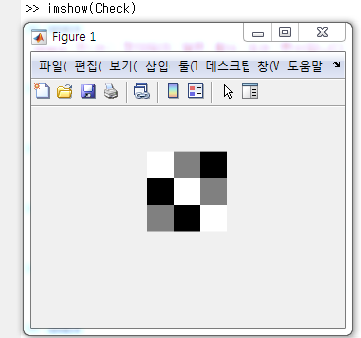
아래는 rank 4에 상응하는 코드이며, 나머지는 결과만 첨부한다.





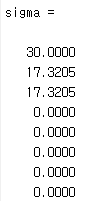
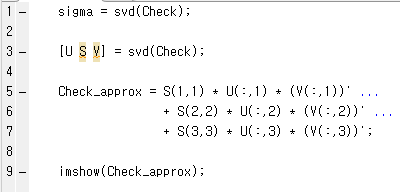
3. (1)

<imshow> 메소드의 입력으로 행렬 Check를 넣는다. 코드와 그 결과는 아래와 같다.



3. (2)

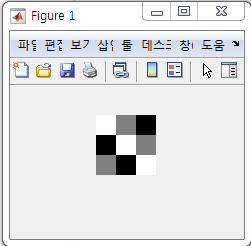
행렬을 SVD했을 때, 특이값이 클수록 그 특이값의 순서에 대응되는 좌특이벡터 및 우특이벡터가 본래 행렬에 기여하는 바가 크다. 이것은 특이값이 0인 순서에 대응되는 특이벡터들은 일종의 dummy 벡터일 뿐, 원래 행렬에 전혀 기여하지 않음을 의미한다. 따라서 행렬 Check를 SVD한 다음, 이것으로부터 0이 아닌 특이값의 개수를 알아내고, 주어진 rank-1 표현을 60번이 아닌 특이값의 개수만큼만 반복함으로써, 본래 행렬을 계산한다. 이 계산에 필요 없는 모든 행렬 요소들은 0으로 제거해서, 행렬 복구를 위한 최소한의 데이터만을 남긴다.



Check를 SVD하면, 특이값이 3개임을 알 수 있다. 오른쪽 그림은 60차원의 특이값 벡터 중 의미 있는 3개의 특이값 원소를 포함한 결과이다. 세 원소를 제외한 나머지는 모두 0이었다. 즉, 행렬 Check의 복구에 필요한 최소의 원소 개수는 (60 + 1 + 60)\*3, 363개이다. float 자료형 1개가 4byte일 때, 전체 이미지 파일은 1452byte 혹은 1.418KB의 데이터가 최소한으로 필요할 것이다.

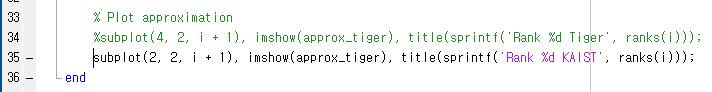
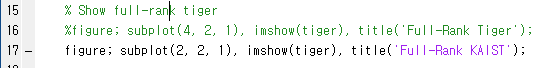
3. (3)

실제로 3.(2)에서 제시된 코드 중, Check\_approx가 바로 3.(2)에서 설명한 방법으로 계산한 행렬이다. 이를 계산하여 <imshow> 메소드로 파일을 열면, 아래와 같이 본래와 완벽히 똑같은 이미지가 등장한다. 결론지으면, SVD는 데이터를 함축적으로 가질 수 있게 해주며, 실제로 함축된 데이터로부터 본래의 상태를 완벽히 복구할 수 있음을 확인하였다.

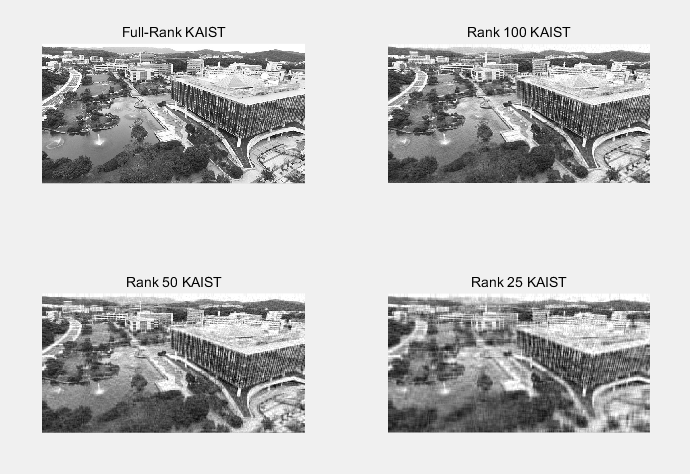


4. (1)

주어진 compress.m 파일 중 일부 코드를 바꾸어 주어진 rank 조건의 이미지를 구한다. 구체적으로 어느 부분의 코드를 고쳤는지는 아래 그림으로 대신한다. 본래 코드를 주석 처리하여, 비교하기 쉽게 하였다.

EMB00000ef84bbcEMB00000ef84bbd

위처럼 코드를 수정하면, 아래와 같은 KAIST의 Low-rank Approximation 사진 3장과 본래 사진 1장이 등장한다.



4. (2)

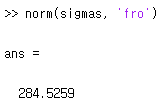
문제에서 주어진 상대오차 DRW00000ef84bc1는 다음과 같다.

DRW00000ef84bc3

모든 행렬 X는 SVD 가능하며, 이것을 DRW00000ef84bc5라 할 때, 다음이 성립한다.

DRW00000ef84bc7

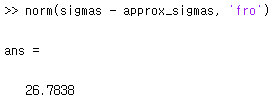
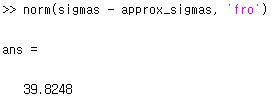
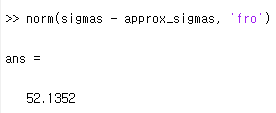
따라서 상대오차 식의 분자는 근사된 행렬의 rank 개의 특이값이 앞에서부터 제거된 대각행렬의 Frobenius Norm일 것이다. 먼저 본래 행렬의 Frobenius Norm을 구한다. 이에 사용된 메소드는 <norm>에 ‘fro’를 추가 입력으로 넣은 것이다.



그 다음으로, 본래 행렬에서 Rank 25, 50, 100에 대해 Low-rank 근사가 된 행렬을 뺀 후. 이들의 Frobenius Norm을 구한다. 각 Low-rank 근사는 코드를 조금 수정하여 구한다.

EMB00000ef84bbd

아래 코드에서 벡터 ranks의 가장 마지막에 등장하는 원소를 오차를 구하고자 하는 rank로 바꾸면 된다. 아래는 각각에 대한 Frobenius Norm을 구한 것이다.



이를 상대오차 식에 대입하면 아래와 같다.

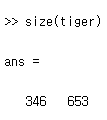
|  |  |  |  |
| --- | --- | --- | --- |
|  | 25 rank | 50 rank | 100 rank |
| 상대 오차 | 0.183235 | 0.139982 | 0.0941348 |

4. (3)

문제에서 제시한 압축률은 다음과 같다.

DRW00000ef84bcd

<size> 메소드로 변수 tiger의 크기를 구한다. (편의를 위해 코드 내의 변수 명은 그대로 두었다.)



곱셈의 결과는 225938이다.

각 rank에 대해 본래 데이터를 복구해내기 위해 필요한 원소의 개수를 구한다.

\* 25 rank: (346 + 1 + 653) \* 25 = 25000

\* 50 rank: (346 + 1 + 653) \* 50 = 50000

\* 100 rank: (346 + 1 + 653) \* 100 = 100000

마지막으로 각 rank에 대하여 압축률을 구한다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 25 rank | 50 rank | 100 rank |
| 압축률 | 0.11065 | 0.22130 | 0.44260 |