

## HW1-Q1.py

```

1 # Peyton Miller, Homework 1
2
3 #Question 1:
4
5 #Asked ChatGPT the following:
6 # "Develop an algorithm in python without numpy that numerically determines the SVD
7 # of a matrix, A.
8 # The algorithm should compute the SVD by
9 # (i) transforming A into a bidiagonal matrix B using Householder reflections,
10 # (ii) determine the QR-factorization of B using Givens rotations to find the SVD of
11 # B,
12 # (iii) determine the SVD of A using the SVD of B. Provide specific comments on each
13 # step above.
14 # You do not need to provide comments on the details of how the Givens rotations or
15 # Householder reflections are applied.
16 # Apply your algorithm to the matrix below, and provide each matrix component of the
17 # decomposition A = UΣVT.
18 # A = [ 1 4 9
19 #       3 2 3
20 #       6 4 5 ]"
21
22 import math
23 import copy
24
25 def matmul(A, B):
26     m, n, p = len(A), len(B), len(B[0])
27     C = [[0.0]*p for _ in range(m)]
28     for i in range(m):
29         for j in range(p):
30             for k in range(n):
31                 C[i][j] += A[i][k] * B[k][j]
32     return C
33
34 def transpose(A):
35     return [list(row) for row in zip(*A)]
36
37 def identity(n):
38     return [[1.0 if i == j else 0.0 for j in range(n)] for i in range(n)]
39
40 def norm(v):
41     return math.sqrt(sum(x*x for x in v))
42
43 def householder_vector(x):
44     v = x[:]
45     alpha = norm(v)
46     if alpha == 0:
47         return v
48     if v[0] >= 0:

```

setting up  
matrix,  $A \in \mathbb{R}^{m \times n}$   
and  $B \in \mathbb{R}^{n \times n}$   
with transpose,  
identity, and norm  
properties

Householder reflections (step 1)  
for  $A = U, B, V^T$   $U, V$  are  
 $H_u(x) = \tilde{x} - 2\tilde{u}\tilde{u}^T$ ,  $\tilde{u} \in \mathbb{R}^n$  bidiagonal

```

45     alpha = -alpha
46     v[0] -= alpha
47     beta = norm(v)
48     return [vi / beta for vi in v]
49
50 def apply_householder_left(A, v, k):
51     for i in range(k, len(A)):
52         s = sum(v[j-k] * A[j][i] for j in range(k, len(A)))
53         for j in range(k, len(A)):
54             A[j][i] -= 2 * v[j-k] * s
55
56 def apply_householder_right(A, v, k):
57     for i in range(len(A)):
58         s = sum(v[j-k] * A[i][j] for j in range(k, len(A[0])))
59         for j in range(k, len(A[0])):
60             A[i][j] -= 2 * v[j-k] * s
61
62
63 def bidiagonalize(A):    step 2: computing  $B = U\Sigma V^T$  using QR
64     A = copy.deepcopy(A)
65     m, n = len(A), len(A[0])
66     U = identity(m)
67     V = identity(n)
68
69     for k in range(min(m, n)):
70         apply left householder
71         x = [A[i][k] for i in range(k, m)]
72         v = householder_vector(x)
73         apply_householder_left(A, v, k)
74         apply_householder_left(U, v, k)
75
76         if k < n - 1:
77             apply left right householder
78             x = [A[k][j] for j in range(k+1, n)]
79             v = householder_vector(x)
80             apply_householder_right(A, v, k+1)
81             apply_householder_right(V, v, k+1)
82
83     return U, A, V     $\Rightarrow A = U\Sigma V^T$ 
84
85
86 def givens(a, b):
87     r = math.hypot(a, b)
88     return a/r, b/r
89
90
91 def svd_bidiagonal(B, iterations=50):
92     n = len(B)
93     U = identity(n)
94     V = identity(n)

```

applying to the left and to the right unit  $A = U\Sigma V^T$

step 2: computing  $B = U\Sigma V^T$  using QR

apply left householder

apply left right householder

$\Rightarrow \beta$  becomes

$B = U_2 \Sigma V^T$

$\uparrow$  bidiagonal

STEP 3:

determine SVD of A using B.

```

95     B = copy.deepcopy(B)
96
97     for _ in range(iterations):
98
99         for i in range(n-1):
100            c, s = givens(B[i][i], B[i+1][i])
101            for j in range(n):
102                t1 = c*B[i][j] + s*B[i+1][j]
103                t2 = -s*B[i][j] + c*B[i+1][j]
104                B[i][j], B[i+1][j] = t1, t2
105
106            t1 = c*U[i][j] + s*U[i+1][j]
107            t2 = -s*U[i][j] + c*U[i+1][j]
108            U[i][j], U[i+1][j] = t1, t2
109
110        for i in range(n-1):
111            c, s = givens(B[i][i], B[i][i+1])
112            for j in range(n):
113                t1 = c*B[j][i] + s*B[j][i+1]
114                t2 = -s*B[j][i] + c*B[j][i+1]
115                B[j][i], B[j][i+1] = t1, t2
116
117            t1 = c*V[j][i] + s*V[j][i+1]
118            t2 = -s*V[j][i] + c*V[j][i+1]
119            V[j][i], V[j][i+1] = t1, t2
120
121 Sigma = [[0.0]*n for _ in range(n)]
122 for i in range(n):
123     Sigma[i][i] = abs(B[i][i])
124
125 return U, Sigma, V
126
127 for lines 95-125:

```

$$\begin{aligned}
A &= U_1 B V_1^T & B &= U_2 \Sigma V_2^T \\
\text{sub } B & & \downarrow & \\
A &= (U_1, U_2) \sum_{i=1}^3 (V_i, V_i^T)
\end{aligned}$$

Applied to A

$$\begin{aligned}
A &= \begin{bmatrix} 1 & 4 & 9 \\ 3 & 2 & 3 \\ 6 & 4 & 5 \end{bmatrix} \Rightarrow V = \begin{bmatrix} -0.3917 & -0.8293 & 0.3975 \\ -0.3481 & 0.5306 & 0.7731 \\ -0.8519 & 0.1793 & -0.9924 \end{bmatrix} \\
\Sigma &= \begin{bmatrix} 13.7986 & 0 & 0 \\ 0 & 5.6949 & 0 \\ 0 & 0 & 0.4583 \end{bmatrix} U = \begin{bmatrix} -0.5654 & 0.16293 & 0.5335 \\ -0.3297 & -0.26293 & 0.5324 \\ -0.7516 & 0.0509 & -0.6577 \end{bmatrix}
\end{aligned}$$

## HW1-Q2.py

```

1 #Question 2
2
3 #Asked ChatGPT the following:
4 #To solve a linear inverse problem (i.e.,  $Ax = b$ ) using the SVD,  $A = U\Sigma V^T$ ,
5 # we typically solve three linear systems for the problem  $x = A^{-1}b = V\Sigma^{-1}U^T b$ :
6 # Solve  $y = U^T b$  for  $y$ .
7 # Solve  $z = \Sigma^{-1}y$ , where  $\Sigma^{-1} = \text{diag}(1/\sigma_i)$  for  $z$ .
8 # Solve  $x = Vz$  for  $x$ .
9 # Develop an algorithm in python without numpy using your SVD derivation that solves
10 # these three systems.
11 # Apply this to the previous problem to find  $x$  with  $A = [1 4 9 3 2 3 6 4 5]$ ,  $b =$ 
12 #  $[18, 10, 19]^T$ 
13
14 def matvec(A, x): defn. matrix A and vector x
15     return [sum(A[i][j] * x[j] for j in range(len(x)))]
16         for i in range(len(A))]
17
18 def transpose(A): also defined A
19     return [list(row) for row in zip(*A)] as invertible
20
21
22 def svd_solve(U, Sigma, V, b, tol=1e-12): A^T A = I
23
24     defn the SVD decomp
25     A = U \Sigma V^T
26
27     UT = transpose(U)
28     y = matvec(UT, b) and y as y = U^T b
29
30     z = [] ← singular values
31     for i in range(len(Sigma)):
32         sigma = Sigma[i][i]
33         if abs(sigma) < tol: z = \Sigma^{-1} y
34             z.append(0.0)
35         else:
36             z.append(y[i] / sigma)
37
38     x = matvec(V, z) → \vec{x} = V \Sigma^{-1} y = V \Sigma^{-1} U^T b
39
40
41     return x
42

```

Apply to,

$$A = \begin{bmatrix} 1 & 4 & 9 \\ 3 & 2 & 3 \\ 6 & 4 & 5 \end{bmatrix}, b = \begin{bmatrix} 18 \\ 10 \\ 19 \end{bmatrix} \Rightarrow \vec{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Question 3:

Asked ChatGPT the following:

1. Construct the SVD in python without numpy of the image, which you can treat as a matrix  $A$ , with the SVD decomposition given by  $A = U\Sigma V^T$

2. Plot the singular values of the image, which should be a decreasing function.

For each image, determine a reasonable location (call this  $k$ ) at which the singular values level off and are no longer decreasing rapidly.

3. Construct two matrices,  $A_3$  and  $A_k$   $A_3 = \sum_{i=1}^3 u_i \sigma_i v_i^T$  and  $A_k = \sum_{i=1}^k u_i \sigma_i v_i^T$  where  $k$  is the cutoff you

found for the image. 4. Visualize and compare  $A_3$  and  $A_k$  to the full image,  $A$ .

Also generate the errors,  $E_3 = A_3 - A$  and  $E_k = A_k - A$  and comment on where the errors are located.

Repeated the same for image 1 and image 2 in blackboard

Image 1:

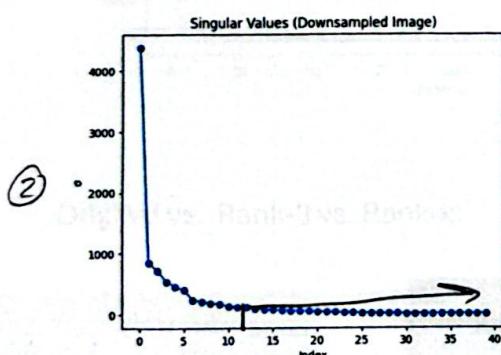


①

$\Rightarrow$  Treated as  $A \in \mathbb{R}^{40 \times 40}$

$$A = U \Sigma V^T \quad \begin{cases} U = U_1, U_2 \\ V = V_1, V_2 \end{cases}$$

Singular Value Graph:



②

③ Rank-3 errors:

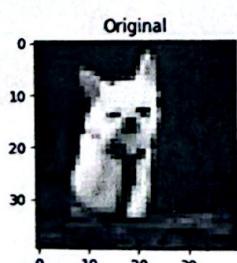
- edges
- facial features
- background

Rank-12 errors:

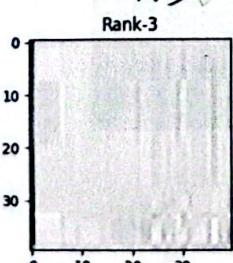
- fur texture
- facial
- background

Original vs. Rank-3 vs. Rank-k

$A_3$



Rank-3



$A_k$

$k=12$  because that's when graph levels

still not as formed as original

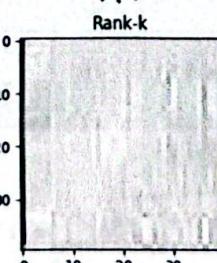


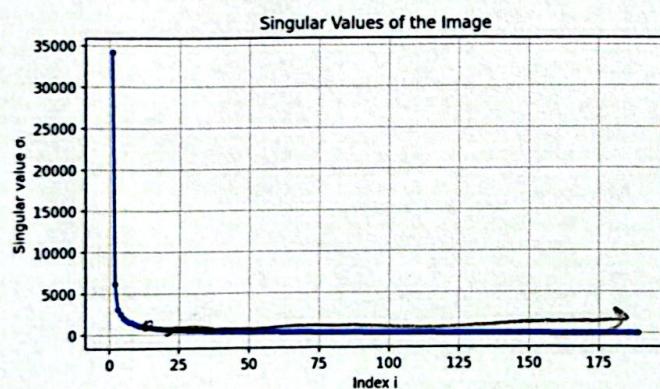
Image 2:

$$\textcircled{1} \quad A \in \mathbb{R}^{40 \times 40}$$



$$\begin{aligned} A &= U \Sigma V^T \quad U = U_1, U_2 \\ \Downarrow \\ A &= U_1 B V_1^T \quad V = V_1, V_2 \end{aligned}$$

Singular Value Graph:

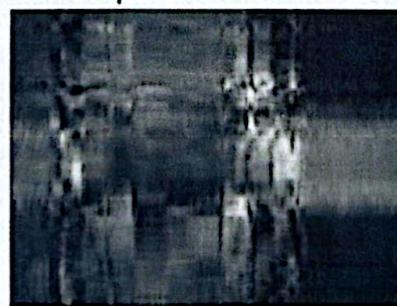


cut off at ~~rank 14~~  
because that is when  
the graph levels

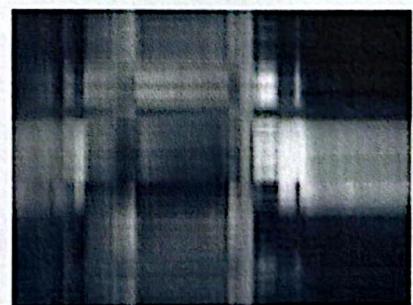
Original vs. Rank-3 vs. Rank-k:



Rank - k



Rank - 3



ERRORS

• dog shape is  
more visible  
but not as  
sharp as  
that of original

• cannot recognize  
subject