SF2526 - Homework 1

Ville Sebastian Olsson, Silpa Soni Nallacheruvu

February 5, 2025

Problem 1

a)

Proof. Let $\mathbf{c}_1, \dots \mathbf{c}_4$ denote the column vectors of A, so that:

$$A = [\mathbf{c}_1\ \mathbf{c}_2\ \mathbf{c}_3\ \mathbf{c}_4] = \begin{bmatrix} 1 & 2 & 2003 & 2005 \\ 2 & 2 & 2002 & 2004 \\ 3 & 2 & 2001 & 2003 \\ 4 & 7 & 7005 & 7012 \end{bmatrix}$$

Observe that $\mathbf{c}_4$ is a linear combination of two other column vectors:

$$\mathbf{c}_4 = \begin{bmatrix} 2005 \\ 2004 \\ 2003 \\ 7012 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 7 \end{bmatrix} + \begin{bmatrix} 2003 \\ 2002 \\ 2001 \\ 7005 \end{bmatrix} = \mathbf{c}_2 + \mathbf{c}_3$$

Since the fourth column is linearly dependent on the remaining columns, the rank of A is at most $4 - 1 = 3$. We can prove that $\text{rank}(A) = 3$ by reducing the matrix to row echelon form:

$$\begin{aligned}
&\operatorname{rank}(A)\\
=&\operatorname{rank}\begin{bmatrix}1 & 2 & 2003 & 0\\ 2 & 2 & 2002 & 0\\ 3 & 2 & 2001 & 0\\ 4 & 7 & 7005 & 0\end{bmatrix}\\
=&\operatorname{rank}\begin{bmatrix}1 & 2 & 2003 & 0\\ 0 & -2 & -2004 & 0\\ 0 & -4 & -4008 & 0\\ 0 & -1 & -1007 & 0\end{bmatrix}\\
=&\operatorname{rank}\begin{bmatrix}1 & 2 & 2003 & 0\\ 0 & 1 & 1002 & 0\\ 0 & 1 & 1002 & 0\\ 0 & 1 & 1007 & 0\end{bmatrix}\\
=&\operatorname{rank}\begin{bmatrix}1 & 2 & 2003 & 0\\ 0 & 1 & 1002 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 5 & 0\end{bmatrix}\\
=&\operatorname{rank}\begin{bmatrix}1 & 2 & 2003 & 0\\ 0 & 1 & 1002 & 0\\ 0 & 0 & 5 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}\\
=&3
\end{aligned}$$

□

b)

In the script hw1_1b.m, we implement and apply Algorithm 1 to the matrix A and print the error after three iterations. The output is shown in Listing 1. The plot of the error for each iteration is provided in Figure 1. As observed, the plot closely resembles the figure in the margin referred to in the question.

```
>> hw1_1b
3rd error: 8.2386e-13
```

Listing 1: The error is $\|A_3\| \approx 8 \cdot 10^{-13}$ which is small.

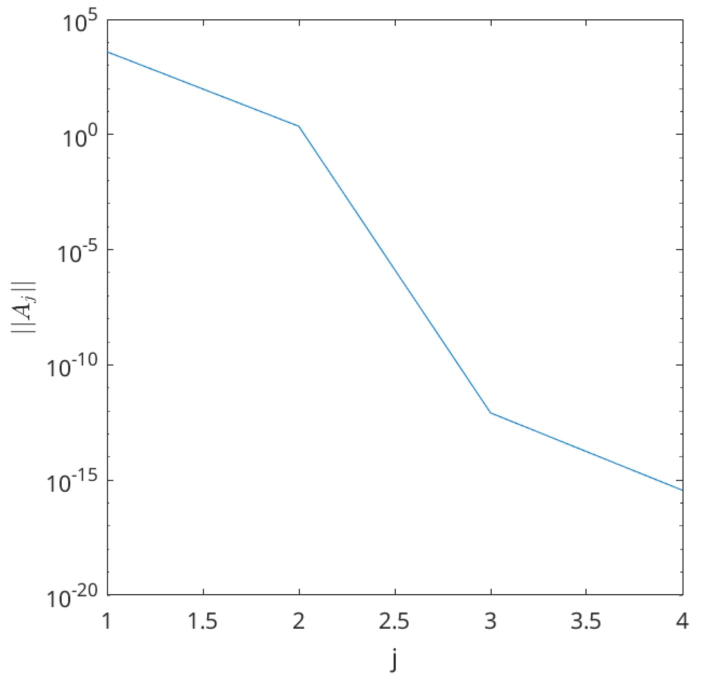

Figure 1: Error $\|A_j\|$ for iteration j.

c)

In `hw1_1c.m`, we apply Algorithm 1 to the larger matrix of size 1000×100. The plot of the error for each iteration is provided in Figure 2.

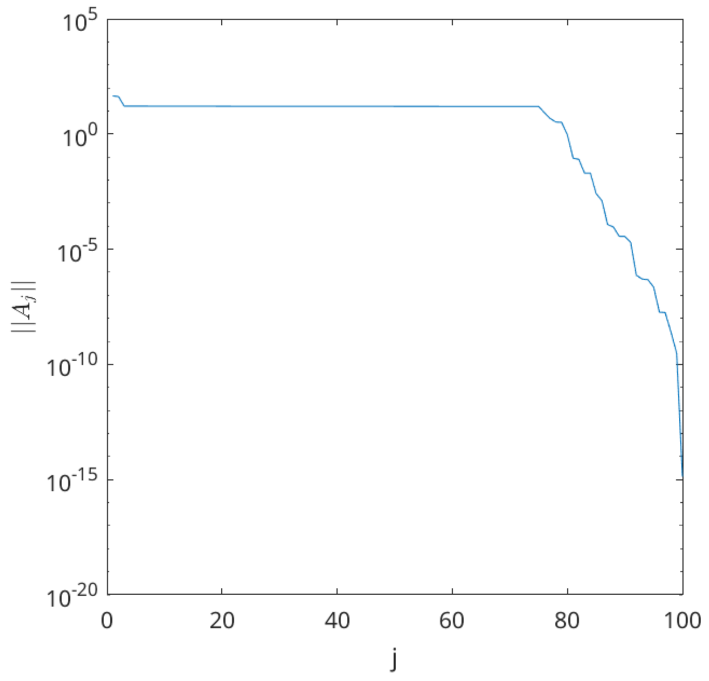

Figure 2: Error $\|A_j\|$ for iteration j.

d)

In `hw1_1d.m`, we apply the greedy variant of Algorithm 1 to the larger matrix. See Figure 3 for a plot of the error for each iteration.

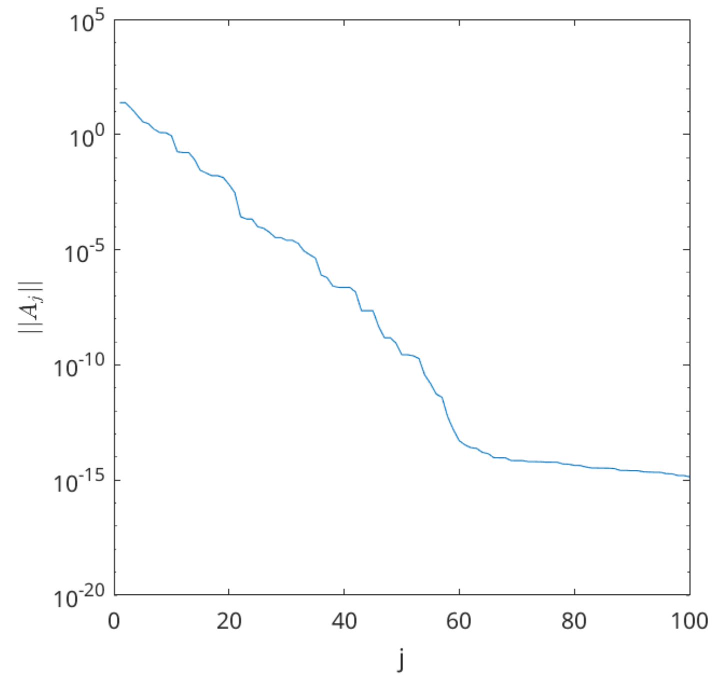

Figure 3: Error $\|A_j\|$ for iteration j.

The greedy algorithm prioritizes the removal of the column vectors with the highest norm during each iteration. This approach maximizes the reduction of the residual error $||A_j||$ in each step, as larger norm vectors contribute more to the total error. Consequently, after $s \leq 100$ iterations, the resulting matrix Q_s more effectively captures the most significant components of the column space of V, leading to a better approximation of a basis for the span of V.

Problem 2

a)

Given $A = \begin{bmatrix} 5 & -1 \\ 5 & 7 \end{bmatrix}$, $S = \begin{bmatrix} \sqrt{80} & 0 \\ 0 & \sqrt{20} \end{bmatrix}$, $V = \beta \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$ and $A = USV^{\top}$,

we first want to determine U and β.

We know that V is an orthogonal matrix, so:

$$\begin{aligned}
&V^\top V = I \\
\Rightarrow &\beta \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}^\top \beta \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\Rightarrow &\beta^2 \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\Rightarrow &\beta^2 \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\Rightarrow &2\beta^2 = 1 \\
\Rightarrow &\beta = \pm\frac{1}{\sqrt{2}}
\end{aligned}$$

S is a diagonal matrix, so its inverse is easy to find:

$$S^{-1} = \begin{bmatrix} \frac{1}{\sqrt{80}} & 0 \\ 0 & \frac{1}{\sqrt{20}} \end{bmatrix}$$

To determine U:

$$\begin{aligned}
&A = USV^\top \\
\Rightarrow &AV = USV^\top V \\
\Rightarrow &AV = USV^{-1}V \\
\Rightarrow &AV = US \\
\Rightarrow &AVS^{-1} = USS^{-1} \\
\Rightarrow &U = AVS^{-1} \\
\Rightarrow &U = \begin{bmatrix} 5 & -1 \\ 5 & 7 \end{bmatrix} \left(\pm\frac{1}{\sqrt{2}}\right) \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{80}} & 0 \\ 0 & \frac{1}{\sqrt{20}} \end{bmatrix} \\
\Rightarrow &U = \left(\pm\frac{1}{\sqrt{2}}\right) \begin{bmatrix} 5 & -1 \\ 5 & 7 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{80}} & -\frac{1}{\sqrt{20}} \\ \frac{1}{\sqrt{80}} & \frac{1}{\sqrt{20}} \end{bmatrix} \\
\Rightarrow &U = \left(\pm\frac{1}{\sqrt{160}}\right) \begin{bmatrix} 5 & -1 \\ 5 & 7 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 1 & 2 \end{bmatrix} \\
\Rightarrow &U = \left(\pm\frac{1}{4\sqrt{10}}\right) \begin{bmatrix} 4 & -12 \\ 12 & 4 \end{bmatrix} \\
\Rightarrow &U = \pm\frac{1}{\sqrt{10}} \begin{bmatrix} 1 & -3 \\ 3 & 1 \end{bmatrix}
\end{aligned}$$

We can verify that U is an orthogonal matrix using MATLAB. See Listing 2.

```
>> U = 1/sqrt(10)*[1 -3 ; 3 1]

U =

    0.3162   -0.9487
    0.9487    0.3162

>> U'*U

ans =

    1.0000         0
         0    1.0000

>> U = -1/sqrt(10)*[1 -3 ; 3 1]

U =

   -0.3162    0.9487
   -0.9487   -0.3162

>> U'*U

ans =

    1.0000         0
         0    1.0000
```

Listing 2: Since $U^\top U = I$, U is orthogonal.

b)

Due to the Eckart-Young theorem, the error $\|A - X\|$ is given by the $(k+1)$th singular value. Since $k = 1$ in this case, the error is equal to the second singular value:

$$\|A - X\| = \sigma_2 = S_{2,2} = \sqrt{20} \approx 4.4721$$

To test this assertion, first we compute the matrix X.

$$\begin{aligned}
&X \\
=\ &\operatorname*{argmin}_{\text{rank}(X)=1} \|A - X\| \\
=&\sum_{i=1}^{1} u_i v_i^\top \sigma_i \\
=&u_1 v_1^\top \sigma_1 \\
=&\left(\pm\frac{1}{\sqrt{10}}\begin{bmatrix}1\\3\end{bmatrix}\right)\left(\pm\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\end{bmatrix}\right)\sqrt{80} \\
=&\frac{1}{\sqrt{10}}\frac{1}{\sqrt{2}}\sqrt{80}\begin{bmatrix}1\\3\end{bmatrix}\begin{bmatrix}1 & 1\end{bmatrix} \\
=&\frac{1}{\sqrt{10}}\frac{1}{\sqrt{2}}\sqrt{80}\begin{bmatrix}1 & 1\\3 & 3\end{bmatrix} \\
=&2\begin{bmatrix}1 & 1\\3 & 3\end{bmatrix} \\
=&\begin{bmatrix}2 & 2\\6 & 6\end{bmatrix}
\end{aligned}$$

Next, we compute the spectral norm error $\|A - X\|$ using MATLAB to ensure that it matches the second singular value, as shown in Listing 3.

```
>> A = [5 -1; 5 7]

A =

     5    -1
     5     7

>> X = [2 2 ; 6 6]

X =

     2     2
     6     6

>> norm(A-X, 2)

ans =

    4.4721
```

Listing 3: $\|A - X\| = \sqrt{20} = \sigma_2$.

Problem 3

a)

Using the Eckart-Young theorem, the best approximation to a matrix A is obtained by selecting the dominant components from its singular value decomposition (SVD).

The SVD of A is given as:

$$A = U\Sigma V^\top,$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix of singular values in non-increasing order.

We want to find the minimum rank k such that $\sigma_{k+1} < 10^{-10}$, where σ_{k+1} is the $(k+1)$th element along the diagonal of the Σ matrix. The best rank-k approximation of A is given by:

$$X = U_k \Sigma_k V_k^\top,$$

where U_k, Σ_k, and V_k correspond to the first k singular values and their associated singular vectors.

Using the script `hw1_3a.m`, we verify that the error $\|A - X\|_2$ is just below 10^{-10}. The resulting matrix has rank $k = 52$. See Listing 4 for the output.

```
>> hw1_3a
Lowest rank: 52
||A-X|| = 7.9965e-11
```

Listing 4: With a $k = 52$ rank approximation of the original matrix A, the resulting matrix X is within a 10^{-10} error of A.

b)

In the script `hw1_3b.m`, we apply Algorithm 1 to the original matrix to iteratively build Q_j and R_j, but we stop iterating once $\|A_j\|_F < 10^{-10}$. Then we perform an SVD of R_k, where k is the final iteration number:

$$R_k = \hat{U}\Sigma V^\top$$

then compute $U = Q\hat{U}$ and finally compute the k-rank approximation $X = U\Sigma V^\top$. Refer to the output in Listing 5.

```
>> hw1_3b
Rank: 54
```

Listing 5: With tolerance 10^{-10}, the resulting low-rank approximation has rank $k = 54$.

Problem 4

The summary of the video quizzes is mentioned below:

Video quiz 1: Low-rank approximation example

The application of low-rank approximation is demonstrated using a sequence of images extracted from the MNIST dataset, specifically four handwritten digit '9' images. Each image is represented as a matrix, where the pixel intensities are simplified to values of 0 (black), 1 (white), or intermediate values (e.g., 0.5 for gray). These matrices are then vectorized by reshaping them into column vectors, which are combined into a single matrix. The rank of this matrix reflects the similarity of the images (e.g., rank one if all images are identical or scalar multiples of each other). The goal is to compute a rank-one approximation of the matrix that minimizes the Frobenius norm of the difference, effectively finding a vectorized image that best represents the set of images under the low-rank constraint.

Video quiz 2a: QR-factorization from Gram-Schmidt

Start with linearly independent vectors $v_1, \ldots, v_p \in \mathbb{R}^n$ and transform them into an orthogonal basis $w_1, \ldots, w_p$ by removing overlapping components using projections of each vector v_i onto the orthogonal vectors computed previously. The Gram-Schmidt process ensures that each new vector is orthogonal to the span of the previously processed ones. Normalize w_i to obtain $q_i = \frac{w_i}{\|w_i\|}$, forming the columns of the orthogonal matrix Q. The coefficients from the projections in Gram-Schmidt process form an upper triangular matrix R. The result is the QR factorization $A = QR$, where Q is orthogonal and R is upper triangular.

Video quiz 2b: Solving linear least squares via QR-factorization

QR-factorization simplifies solving the linear least squares problem $\min \|Ax - b\|_2$ by substituting $A = QR$ into the normal equations $A'Ax = A'b$. This results in a system with an upper triangular matrix R, which is solved efficiently using backward substitution. This approach is more stable and computationally efficient than directly solving the normal equations and is the method used by MATLAB's backslash operator for rectangular matrices.

Video quiz 3a: SVD introduction and properties

The video gives an introduction to singular value decomposition (SVD) and its properties. SVD enables us to factorize a matrix into two orthogonal matrices and one rectangular diagonal matrix. The diagonal matrix consists of non-negative so-called singular values which appear in non-increasing order, and these values can be computed from the eigenvalues of a particular transformation of the original matrix. SVD can be used to find a solution to a linear least squares problem as an alternative to QR decomposition. The Eckart-Young theorem implies that SVD yields the best low-rank approximation (i.e. better than QR-decomposition).

Video quiz 3b: SVD and the Eckart-Young theorem

A matrix can always be SVD-factorized. Furthermore, a matrix can be expressed as a sum of rank-1 matrices constructed from the product of each singular value (σ_i) and vectors of the two orthogonal matrices ($\mathbf{u}_i$, $\mathbf{v}_i^\top$). The Eckart-Young theorem states that the best rank-k approximation is given by the first k terms in this sum. Moreover, the $(k+1)$th singular value the error of the approximation.

Video quiz 4a: Index vectors

Given a matrix $A \in \mathbb{R}^{m \times n}$, an index vector is a vector of indices used to create a matrix consisting of columns of A. The complement $\bar{J}$ of an index vector J with respect to a matrix A is a vector such that $J \cup \bar{J} = \{1, \dots, n\}$. If an index vector is used to select columns from an identity matrix, the resulting matrix is called a permutation matrix.

Video quiz 4b: Randomized SVD

Randomized SVD is an algorithm for obtaining a low-rank approximation and consists of two stages, A and B. In stage A, we generate linearly independent Gaussian vectors and use them together with the orignal matrix to construct an approximate basis for the column space of the matrix. Stage B is a deterministic phase, where we perform a singular value decomposition on a transformation of the resulting approximation from stage A. In stage A, it is often helpful to draw 5-10 extra Gaussian vectors, as it may improve the approximation considerably.

Video quiz 5: Introduction to interpolatory decompositions

ID expresses A as $A = CZ$, where C is formed from a subset of columns of A and Z contains interpolation weights, typically bounded by 1. It provides a low-rank approximation, but is generally less optimal than SVD-based approximations, like those guaranteed by the Eckart-Young theorem. However, ID has the advantage of selecting actual columns, offering interpretability.

Problem 5

c)

In the script `hw1_5c.m`, we construct matrices A, u, v, and $A - uv^\top$, then output the rank of $uv^\top$. See output in Listing 6.

```
>> hw1_5c
Rank of u*v': 1
||A-uv'||: 17686.3437
Relative error of ||A-uv'||/||A||: 0.016575
```

Listing 6: The resulting rank is 1. The norm of the difference matrix is also displayed.

A relative error of 0.0166 signifies that the rank-1 approximation $uv^\top$ is highly accurate, preserving 98.34% of the overall magnitude of A while accounting for only 1.66% of the discrepancy. This indicates that the rank-1 model provides an efficient and effective representation of A. This result aligns with the observation that there is almost no movement in the video, making the low-rank structure a natural fit for the data.

d)

We follow the same approach as in Task 3b, applying a low-rank approximation method. The resulting low-rank approximation yields a rank of $\text{rank}(X) = 2$ where X is the low rank approximation of A. Refer to the script `hw1_5d.m` and the corresponding output in Listing 7.

```
>> hw1_5d
tol: 17686.3437
Rank: 2
||A-X|| = 12484.352
```

Listing 7: With a tolerance of $\text{tol} = \|A - uv^\top\|$, the numerical rank of the low-rank approximation becomes 2.

e)

We apply both algorithms on the testbild matrix with a few different tolerance levels. We re-run both algorithms 3 times and take the mean of the resulting CPU times in an effort to decrease variance of the result. Refer to the script `hw1_5e.m`. The exact elapsed time and tolerance levels are shown in Listing 8.

Figure 4 shows that for a sufficiently high tolerance level ($> 2 \cdot 10^4$), our implementation performs better than MATLAB's `svd(A,0)` if we rely on the Frobenius norm in each iteration j to calculate $\|A_j\|$. This is because a high tolerance means that fewer iterations are necessary and the resulting Q_j and R_j matrices become small and cheap to decompose. If we rely on the spectral norm, the performance of our algorithm is consistently worse, suggesting that the spectral norm is more expensive to compute. With a different matrix, our implementation may outperform `svd(A,0)` regardless of norm.

```
>> hw1_5e
Tolerance levels:       10000   20000   40000   80000
Naive SVD:              0.52852        0.552      0.52949      0.52355
Approx SVD, Euclidean:  2.936      0.78243      0.77988      0.77205
Approx SVD, Frobenius:  3.4453      0.60897      0.24063      0.24205
```

Listing 8: CPU time of the algorithms for each tolerance level.

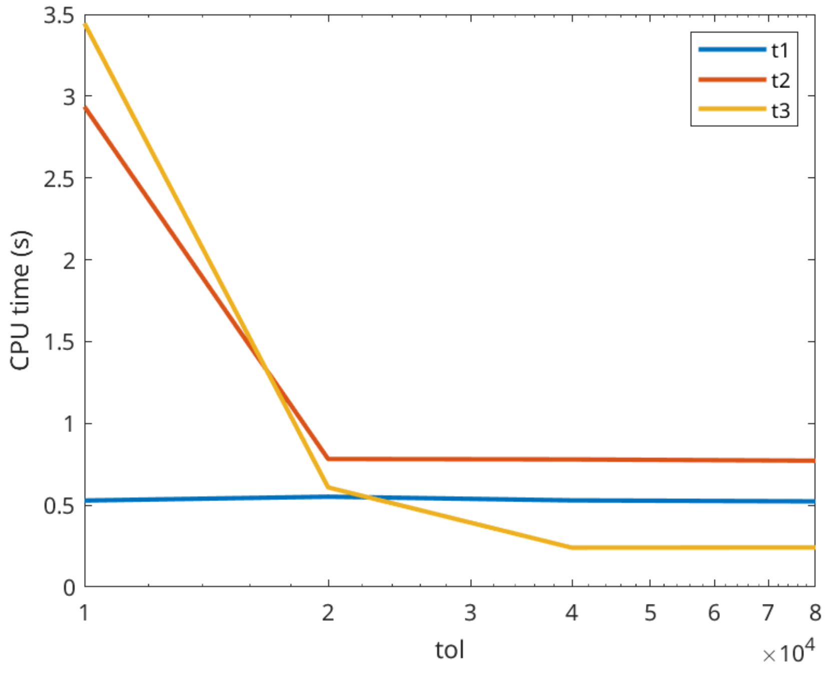

Figure 4: Comparison of the time it takes to run the two algorithms depending on tolerance level (tol). `t1` corresponds to MATLAB's `svd(A,0)`. `t2` corresponds to our implementation using the spectral norm. `t3` corresponds to our implementation using the Frobenius norm.

f)

By extracting $\tilde{A}_1 = u_1\sigma_1 v_1^\top$ from `svd(A,0)`, we obtain a lower approximation error $\|A - \tilde{A}_1\|$. It is shown in the output in Listing 9. Refer to the script `hw1_5f.m` for the implementation of the partial SVD to obtain $\tilde{A}$.

```
>> hw1_5f
||A-uv'|| = 17686.3437
||A-X|| = 12484.352
||A-A1|| = 3565.7688
```

Listing 9: First line: $\|A - uv^\top\|$ from c). Second line: $\|A - X\|$ from d). Third line: $\|A - \tilde{A}_1\|$.

Since $\|A-\tilde{A}_1\| < \|A-X\| < \|A-uv^\top\|$, $\tilde{A}_1$ can be considered a better rank-one approximation than X, which in turn is better than $uv^\top$. This aligns with the fact that the SVD provides a more optimal low-rank approximation, as it minimizes the reconstruction error across all singular components, compared to forming a rank-1 approximation by taking the first column of A as a basis vector and scaling it with a row vector of ones and by forming a rank-1 approximation from Gram-Schimdt column elimination.

Problem 6

a)

Let A denote the matrix containing all 56 images as column vectors. We apply the greedy version of Algorithm 1 to A, making sure to record the error $\|A_j\|$ in each iteration j. Refer to Figure 5 for a plot of the error against each iteration and `hw1_6a.m` for the implementation of the greedy version of Algorithm 1.

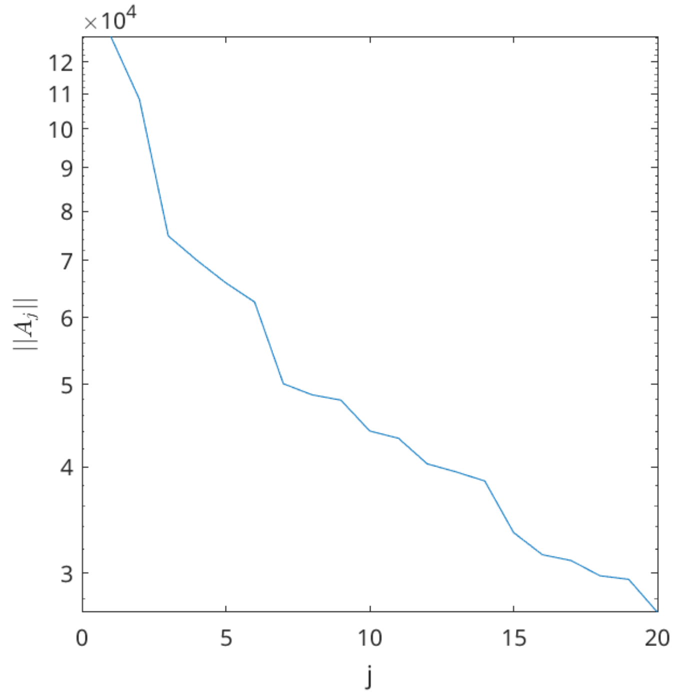

Figure 5: Error $\|A_j\|$ after iteration j.

b)

For each of $k = 1 \ldots, 20$, we compute a k-rank approximation of A using the greedy version of Algorithm 1. For each k, we then compute the norm of the difference between the first snapshot of the 56 snapshots and the corresponding image in the k-approximation.

Refer to `hw1_6b.m` and Figure 6 for a plot of the norm for each value of k. We see that at $k = 9$, the norm $\|A(:,1) - (QR)(:,1)\|$ decreases sharply to almost zero. This indicates that the greedy version of Algorithm 1 picks column 1 during the 9th iteration, making it part of the orthogonal basis of Q.

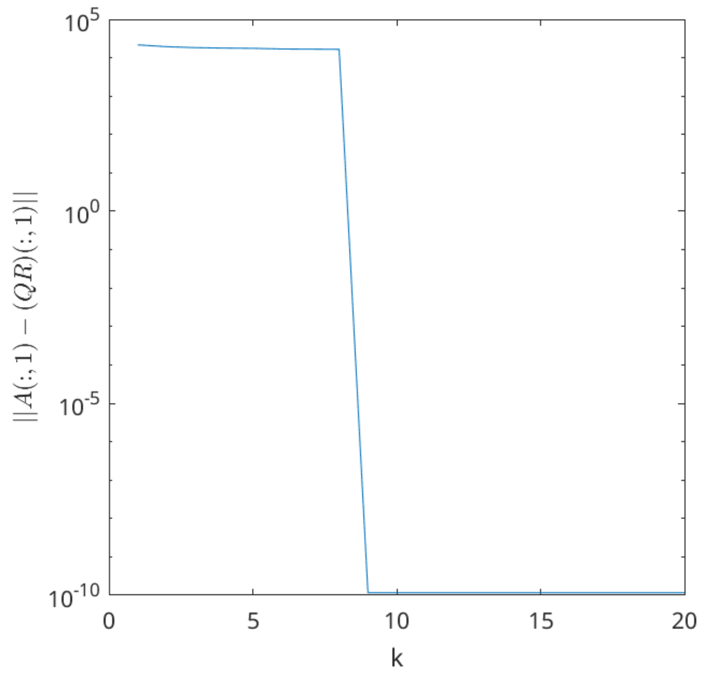

Figure 6: Norm of the difference between the first of the 56 original snapshots and the corresponding image in the k-rank approximation.

c)

In `hw1_6c.m`, we read the snapshots into a matrix A, apply the greedy algorithm with each value of k, construct the k-rank approximation matrix, and reshape this matrix into a sequence of images. The first image in the sequence of images resulting from each k-rank approximation is visualized in Figure 7.

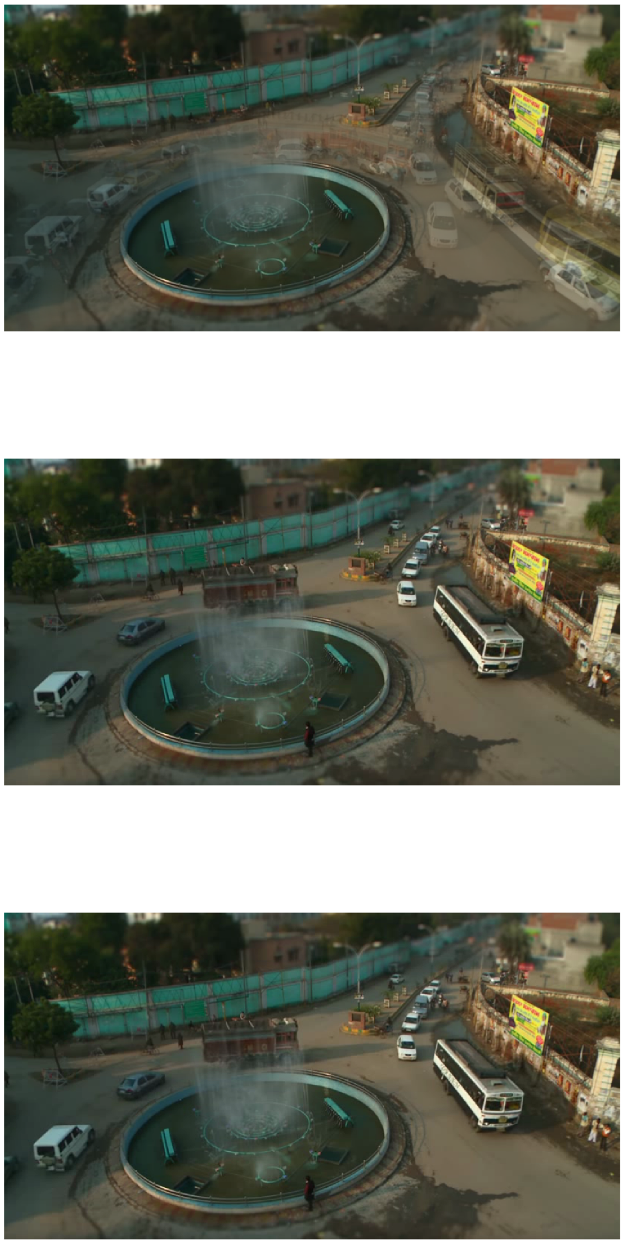

Figure 7: Visualization of the first image of the k-rank approximations for $k = 5, 10, 20$ (ordered from top to bottom).

d)

In the script `hw1_6d.m`, we create two AVI video files, namely `out1.avi` and `out3.avi` based on $k = 1$ and $k = 3$.

With $k = 1$, the 15th image is selected by the greedy algorithm, as it happens to have the largest norm. Hence, every frame in the video is a scaled version of the 15th image. The only difference between the frames is the brightness: the brightness of each frame depends on the overall brightness of the corresponding image. With $k = 3$, the video flips between three different frames, constituting a very rough approximation of the animation occurring in the original video.

Problem 7

a)

Refer to the script `hw1_7a.m` for the implementation of the randomised SVD and the error is shown in Figure 8 for different values of the oversampling parameter $s = 0, \dots, 25$ with fixed rank $k = 5$. The error fluctuates significantly at the beginning but stabilizes to a lower value for $s \geq 5$.

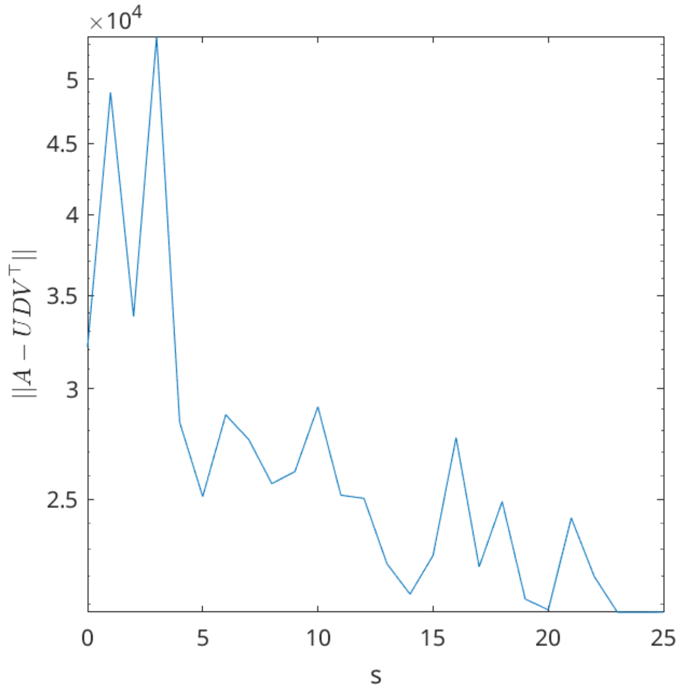

Figure 8: Visualization of the 5-rank approximate error using randomised SVD for the oversampling parameter s ranging from 0 to 25.

b)

We run the greedy version of Algorithm 1 on the matrix repeatedly, measure the CPU-time, and calculate the mean. We do the same for the randomized SVD algorithm in `hw1_7b.m`. The result is shown in Listing 10. We see that the randomized SVD algorithm is more than twice as fast as the greedy version of Algorithm 1.

```
>> hw1_7b
Iteration 1/10: Greedy:3.67106, RandomSVD:1.26306
Iteration 2/10: Greedy:4.72836, RandomSVD:1.46553
Iteration 3/10: Greedy:4.36652, RandomSVD:1.90553
Iteration 4/10: Greedy:3.29242, RandomSVD:1.40032
Iteration 5/10: Greedy:3.32522, RandomSVD:1.42522
Iteration 6/10: Greedy:3.2677,  RandomSVD:1.41
Iteration 7/10: Greedy:3.31554, RandomSVD:1.36585
Iteration 8/10: Greedy:3.29641, RandomSVD:1.33851
Iteration 9/10: Greedy:3.67243, RandomSVD:1.27483
Iteration 10/10:Greedy:3.30892, RandomSVD:1.53967
Greedy Algorithm 1: 3.6245
Randomized SVD:     1.4389
```

Listing 10: The final two lines in the output shows the mean CPU time of running the greedy version of Algorithm 1 and randomized SVD, respectively, with $P = 10$ trials.

Problem 8

a)

We calculate the best k-rank approximation using both partial SVD approximation and ID-approximation in `hw1_8a.m`. The relative errors is shown in Figure 9 and the smallest rank where the error falls below 25% is identified for both methods in Listing 11.

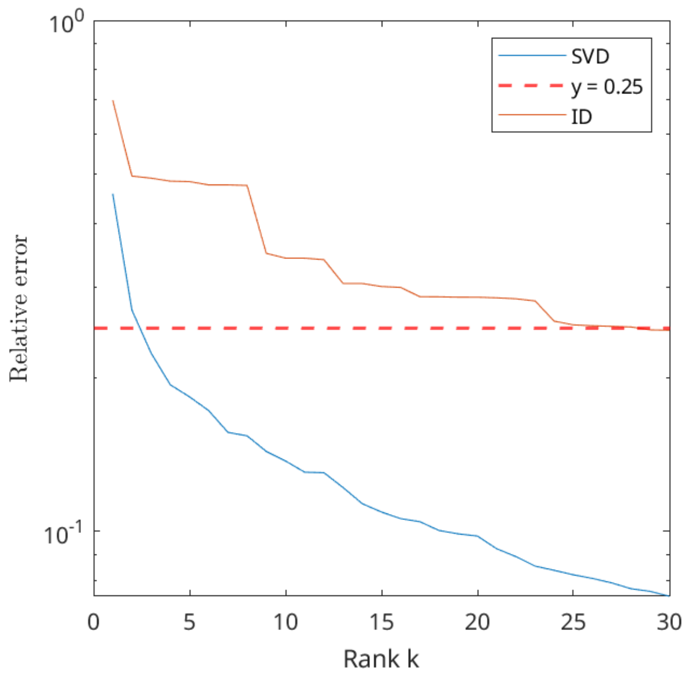

Figure 9: Visualization of relative error in the ID-approximation and partial SVD approximation for rank k ranging from 1 to 30

```
>> hw1_8a
Partial SVD rank: 3
ID rank:          29
```

Listing 11: Rank at relative error 0.25.

As observed, the partial SVD approximation reaches a relative error of 0.25 at rank 3, whereas the ID approximation reaches it at a rank of 29. This suggests that partial SVD provides a significantly more efficient low-rank approximation and ID approximation requires a much higher rank to achieve the same level of accuracy.

b)

As shown in Figure 10, the first three singular images of SVD give an ON basis for the first three images of the original matrix. The first three ID vectors are the three most representative images from the original dataset, meaning they closely resemble the typical images in the collection. These vectors can be used to reconstruct all 6000 images in A. Specifically, the three columns in C are chosen such that $\|A - CZ\|$ is minimized as shown in `hw1_8b.m`.

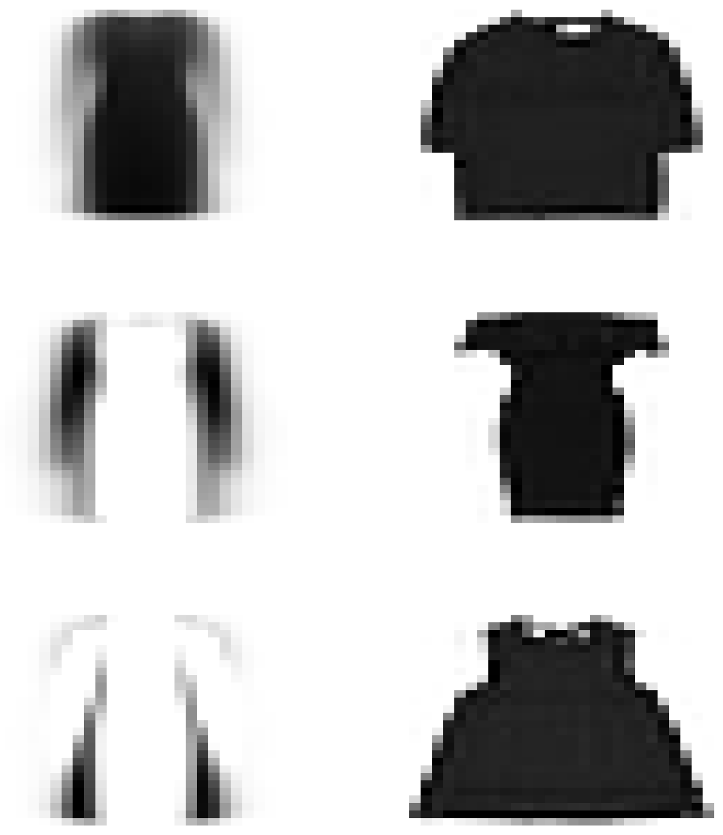

Figure 10: Images from the `item3` image set. Left: the first three singular vectors. Right: the first three ID vectors.

The advantage of ID compared to SVD is that it yields specific, representative elements of the dataset, offering more interpretability and robustness to outliers. Whereas, SVD focuses on global patterns, providing a smooth, holistic representation of the dataset, lacking interpretability in the context of the original data.

The disadvantage of ID compared to SVD is that ID generally does not yield the best low-rank approximation in terms of minimizing the spectral norm of the difference.