香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Project: SVD and Application

Wan Zhenglin ID:121090525

Kong Chuihan ID:122090240

Liu Ziche ID:122090363

Yao Siqi ID:122090664

Contents

1 Introduction **3**

2 Sub-Problem 1: Singular value decomposition using two-phase procedure **3**

2.1 Background . . . 3

2.2 Approaches . . . 3

2.2.1 Phase I:Bidiagonalization . . . 3

2.2.2 Phase II-A:QR Algorithm for Bidiagonal Matrix . . . 4

2.2.3 Phase II-B:Iterative QR and Cholesky Factorizations . . . 4

2.3 Results and Observations . . . 4

2.3.1 Phase I Observations: . . . 4

2.3.2 True results: . . . 5

2.3.3 Phase II-A Results: . . . 5

2.3.4 Phase II-B Results: . . . 6

2.3.5 Conclusions: . . . 6

3 Sub-Problem 2: Application- Deblurring **6**

3.1 Background . . . 6

3.1.1 Blurring . . . 6

3.1.2 Deblurring . . . 7

3.2 New approach:Pseudo Inverse Using **Singular value decomposition** . . . 7

3.3 Test image and Visualization . . . 9

3.4 Two algorithm comparison . . . 10

4 Sub-Problem 3: Power Iterations and Video Background Extraction **10**

4.1 Background . . . 10

4.2 video matrix construction . . . 10

4.3 singular value decomposition of video matrix . . . 12

4.4 Algorithm Comparison . . . 13

4.5 convergence . . . 16

4.6 Additional Colored Reconstruction . . . 16

1 INTRODUCTION

Singular Value Decomposition is instrumental in numerous data science applications, particularly in the domain of image processing. The primary objective of this project is to study different algorithms for computing SVDs and applying these decompositions in various applications.

Through this project, our goal is to deepen our understanding of numerical methods from a theoretical perspective and to gain practical experience in applying these techniques to address real-world challenges.

2 SUB-PROBLEM 1: SINGULAR VALUE DECOMPOSITION USING TWO-PHASE PROCEDURE

2.1 BACKGROUND

The 1st problem aims at exploring numerical methods for computing SVDs. The specific challenge is to implement a two-phase algorithm for SVD, which is particularly focused on matrices that are not necessarily square, and to do so in a manner that is computationally efficient and stable.

2.2 APPROACHES

2.2.1 PHASE I:BIDIAGONALIZATION

Phase I of the problem involves the bidiagonalization of general matrix $A \in R^{m\times n}$ using Householder reflections. This process transforms matrix A into a bidiagonal matrix B , where B has non-zero elements only on the diagonal and the superdiagonal.

Bidiagonalization Procedure: First we perform left Householder transformations. For each column j from 1 to $min(m, n)$,a Householder matrix $H_{L,j}$ is constructed to zero all elements of column j below the diagonal. This transformation updates the matrix as $A := H_{L,j}A$.

Then we perform right Householder transformations. For each column j from 1 to $min(m, n) - 1$,a Householder matrix $H_{R,j}$ is constructed to zero all elements of column j below the diagonal. This transformation updates the matrix as $A := AH_{R,j}$.

Finally, the entire matrix is transformed into the desired bidiagonal form B.The product of all left Householder matrices gives an orthogonal matrix U, and the product of all right Householder matrices gives another orthogonal matrix V,such that $A = UBV^T$.

2.2.2 PHASE II-A:QR ALGORITHM FOR BIDIAGONAL MATRIX

Phase II-A involves eigen decomposition of B^TB to compute the singular value decomposition of bidiagonal matrix B obtained from Phase I.

Iterative QR Algorithm:

The QR algorithm is applied iteratively to the matrix B^TB, where B is the bidiagonal matrix obtained from Phase I. In each iteration, the following steps are performed:

1.Compute the QR decomposition of $B^TB - \mu I$ to obtain Q_k and R_k.

2.Update the matrix B^TB as $B^TB = R_kQ_k + \mu I$.

3.Apply the Wilkinson shift for the next iteration using the new B^TB.

Convergence Criteria:

The stopping criterion for the iterative process can be a predefined tolerance level for the off-diagonal elements. For example, if the absolute values of the off-diagonal elements are less than 10^{-12} , the matrix is considered to have converged to a diagonal form.

2.2.3 PHASE II-B:ITERATIVE QR AND CHOLESKY FACTORIZATIONS

Starting with $X^0 = B$, the algorithm proceeds as follows:

1.QR Factorization: Compute the QR factorization of $(X^k)^T$,yielding Q_kR_k.

2.Cholesky Decomposition: Factorize $R_kR_k^T$ using Cholesky decomposition to obtain $L_kL_k^T$.

3.Matrix update:Set $X^{k+1} = L_k^T$.

4.Convergence Check: Continue until the off-diagonal elements of X^k are below a specified tolerance.

5.Extraction of Singular Values: The diagonal elements of the converged X^k give the squared singular values of A.

2.3 RESULTS AND OBSERVATIONS

The two-phase SVD algorithm was applied to a 3×3 matrix $A_{\text{example}} = \begin{bmatrix} 1 & 2 & 4 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$.The algorithm's implementation in Python successfully decomposed A_{example} into its SVD components U, Σ and V^T.

2.3.1 PHASE I OBSERVATIONS:

The Householder bidiagonalization correctly reduced A_{example} to bidiagonal form, as evidenced by the intermediate matrix B. The resulting orthogonal matrices U and V were verified to be accurate through orthogonality checks.

$$U = \begin{bmatrix} -0.12309149 & 0.94585106 & -0.30035689 \\ -0.49236596 & 0.20457495 & 0.84600523 \\ -0.86164044 & -0.25202155 & -0.44052343 \end{bmatrix},$$

$$B = \begin{bmatrix} -8.12403840 & 1.47530172 \times 10^{1} & -7.32500164 \times 10^{-17} \\ -3.13971404 \times 10^{-16} & -2.66705657 & 1.10277994 \\ -1.24245765 \times 10^{-16} & 6.18127670 \times 10^{-17} & -0.138457683 \end{bmatrix},$$

$$V^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.65079137 & -0.7592566 \\ 0 & -0.7592566 & 0.65079137 \end{bmatrix}.$$

2.3.2 TRUE RESULTS:

The actual SVD of A_{example} can be computed:

$$\begin{bmatrix} 1 & 2 & 4 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 0.253 & 0.934 & 0.250 \\ 0.516 & 0.089 & -0.852 \\ 0.818 & -0.345 & 0.460 \end{bmatrix} \begin{bmatrix} 17.005 & 0 & 0 \\ 0 & 1.681 & 0 \\ 0 & 0 & 0.105 \end{bmatrix} \begin{bmatrix} 0.473 & -0.670 & 0.572 \\ 0.566 & -0.266 & -0.780 \\ 0.675 & 0.693 & 0.253 \end{bmatrix}^T$$

2.3.3 PHASE II-A RESULTS:

The QR algorithm with Wilkinson shift led to a diagonal matrix with entries very close to the expected singular values of A_{example}, , which aligns with the theoretical expectations for the given matrix. The corresponding U and V matrices were also computed.

$$U_{\text{phase\_II\_A}} = \begin{bmatrix} 0.25316197 & 0.93445426 & 0.25040819 \\ 0.51592664 & 0.0885498 & -0.8520438 \\ 0.81836955 & -0.34489734 & 0.4596924 \end{bmatrix},$$

$$S_{\text{phase\_II\_A}} = \begin{bmatrix} 17.00483314 & 0 & 0 \\ 0 & 1.68066389 & 0 \\ 0 & 0 & 0.10497068 \end{bmatrix},$$

$$V_{\text{phase\_II\_A}} = \begin{bmatrix} 0.47312757 & -0.66975196 & 0.57234833 \\ 0.56648092 & -0.26627646 & -0.77986936 \\ 0.67472192 & 0.6932021 & 0.25341898 \end{bmatrix}.$$

2.3.4 PHASE II-B RESULTS:

The iterative QR and Cholesky method produced singular values that closely matched those from Phase II-A. The orthogonal matrices U and V^T refined in Phase II-B were consistent with the orthogonal matrices from Phase II-A within numerical tolerance.

$$U_{\text{phase\_II\_B}} = \begin{bmatrix} -0.11587509 & -0.96851234 & 0.22035609 \\ 0.45430054 & -0.24896356 & -0.85535265 \\ 0.88328024 & 0.00099382 & 0.46884436 \end{bmatrix},$$

$$S_{\text{phase\_II\_B}} = \begin{bmatrix} 17.0032235 & 0 & 0 \\ 0 & 1.68081351 & 0 \\ 0 & 0 & 0.10497127 \end{bmatrix},$$

$$V_{\text{phase\_II\_B}} = \begin{bmatrix} -0.48236903 & -0.665069 & 0.57009065 \\ -0.57007249 & -0.25579192 & -0.78076107 \\ -0.66508457 & 0.70160795 & 0.25575144 \end{bmatrix}.$$

2.3.5 CONCLUSIONS:

The implemented algorithms were successful in computing the SVD of the given matrix. The Householder bidiagonalization effectively reduced the matrix to a form suitable for phase II algorithms. The QR algorithm with Wilkinson shift and the subsequent Cholesky decomposition proved to be effective for this purpose. The methodology could be applied to any matrix with full rank, and the stopping criteria ensured computational efficiency.

3 SUB-PROBLEM 2: APPLICATION- DEBLURRING

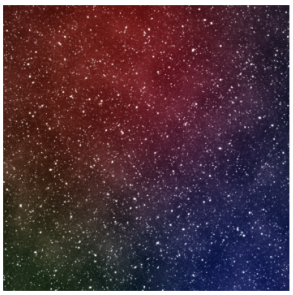
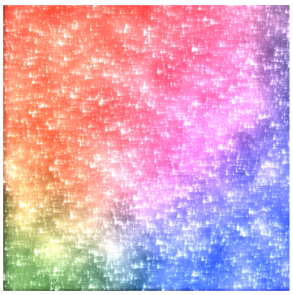
3.1 BACKGROUND

3.1.1 BLURRING

The generalized image blurring problem can be formulated as follows:

$$A_l X A_r = B \tag{1}$$

where A_l and A_r are two blurring kernels. Here we simply choose the kernels the same as in exercise 4. Basically, for the image data there are three dimensions, and the last dimension represents different channels. We perform such blur operation to each channel of the image data. Here we randomly choose three images from the test data folder. The blur effect is shown below.

-  
-  
-  

The right-hand side correspond to the blurred image.

3.1.2 DEBLURRING

For deblurring stage, given the blurred array B, we can deblur B using the natural approach:

$$X_{recovered} = A_l^{-1} B A_r^{-1} \tag{2}$$

But, in application, the kernel matrix A_l and A_r may be **singular** or **bad conditioning**! After our experiment, if we simply use the inverse operation to deblur when the kernel has large conditional number, the deblur effect is tremendously poor.(precisely, will return pure grey image)

3.2 NEW APPROACH:PSEUDO INVERSE USING SINGULAR VALUE DECOMPOSITION

In linear algebra, we can say judge matrix by th e three equivalent argument below:

- Singular(the number of zero singular value is 1)
- rank-deficient

- 0 is the eigenvalue of it(for square matrix)
- do not have inverse matrix.

Combined with the four point of view, we can intuitively say that the existance of 0-singular value make it impossible to invert one matrix. Because if we take the square matrix as example(in our project the kernel are all square matrix),if we apply SVD to square matrix A, then

$$A = U\Sigma V^T \tag{3}$$

$$A^{-1} = V\Sigma^{-1}U^T \tag{4}$$

Where Σ^{-1} will take the inverse of each diagonal element of Σ, if 0-singular value exists, then the outcome will be infinity, causing disastrous deblurring outcome. Thin SVD can address this issue. In thin SVD:

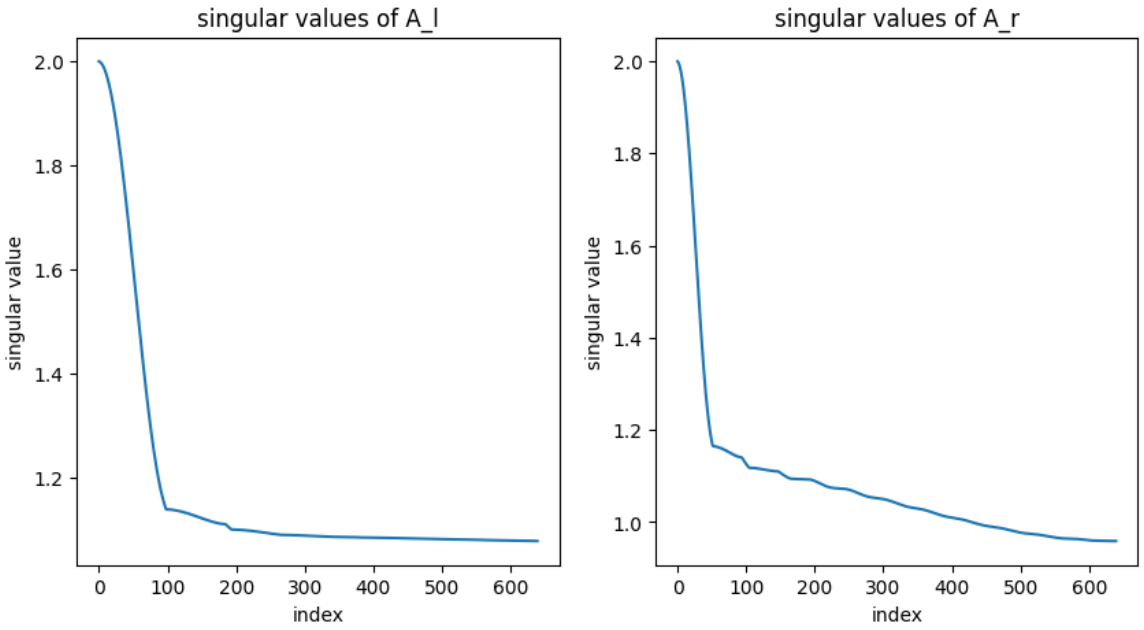
$$A = \hat{U}\hat{\Sigma}\hat{V}^T \tag{5}$$

Where $\hat{\Sigma}$ cuts all the 0-singular value and reserve the positive singular value, and correspondingly cut the singular vectors of U and V. After doing this, take inverse of Σ become possible. Hence, the pseodo-inverse of matrix A can be represented as follows:

$$A^{+} = \hat{V}\hat{\Sigma}^{-1}\hat{U}^T \tag{6}$$

Furthermore, borrowing inspiration from pseudo-inverse, we can design the **Truncated SVD** to give the approximation of original matrix. Precisely, thin SVD cuts all the 0 elements in Σ, so truncated SVD further cuts the small singular values of Σ and corresponding singular vectors in U and V. This is reasonable because the small singular value pairs represent the original matrix in a small degree. And the pseudo-inverse is the same as above. Additionally, we will reserve the $mlargestsingularvalue$ of our kernel matrix, and we will test different values of l and the performance of deblurring.

Firstly we plot the singular values using algorithms from Sub-problem 1:



From the plot:

- For kernel A_l we can set $l_{truncated}$ to be approximately 100. In our project we test three values: 50,100,200.
- For kernel A_r we can set $r_{truncated}$ to be approximately 300, we test three values: 100,300,400.

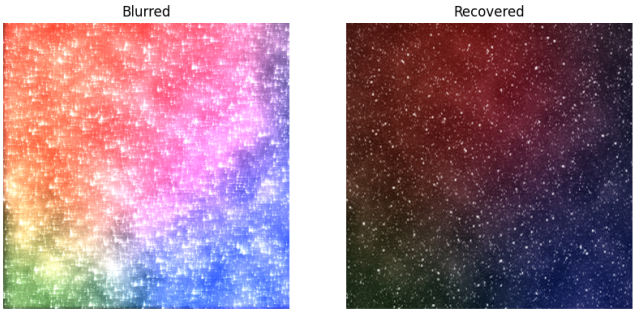
Using PSNR as performance metric, the comparison outcome is:

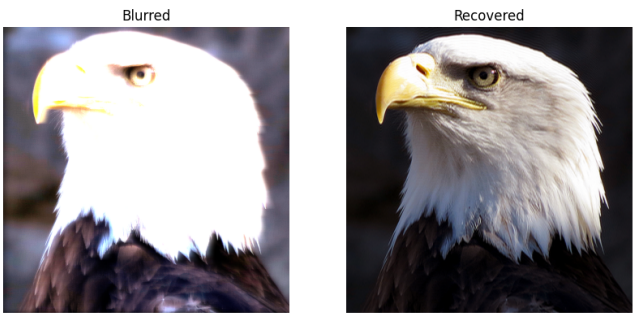
l_trun	r_trun	PSNR
50	100	20.649
50	300	21.157
50	400	21.218
100	100	21.293
100	300	22.515
100	400	22.683
200	100	22.077
200	300	25.106
200	400	25.696

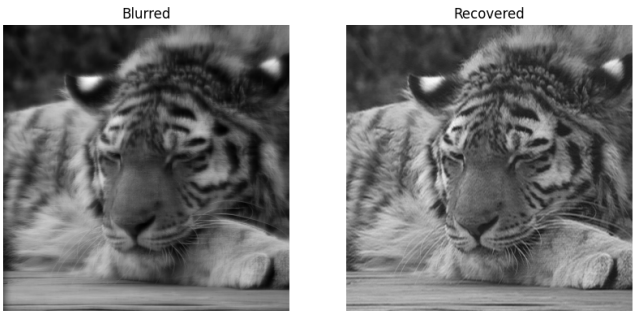
We can see that the singular value of A_r plays a determinant role in the recover performance, since the omitted singular value of A_r is not very small.

3.3 TEST IMAGE AND VISUALIZATION

Here we use the parameter with the highest PSNR, and use our truncated SVD to recover three different images the same as the blur stage we use.

- 

- 

- 

3.4 TWO ALGORITHM COMPARISON

Since the bidiagonalize in sub-Prolem 1 is quite time-consuming in large-scale matrix, we use a approximate version of bidiagonalize, and we test the performance and speed of the two methods. We perform two methods to matrix A_l and compute the norm of difference:$\|U\Sigma V^T - X\|_2$ to verify the accuracy of singular decomposition algorithm, and we also record the time comsumed. The test outcome is:

- Using method A, Time used: **88.14** seconds. Recover difference norm is **4.141444449889223e-13**
- Using method B, Time used: **102.12** seconds. Recover difference norm is **7.694072437602368e-13**

We can see that the method **A** gives a better accuracy and also less time-consuming.

4 SUB-PROBLEM 3: POWER ITERATIONS AND VIDEO BACKGROUND EXTRACTION

4.1 BACKGROUND

In this part of the project, we want to utilize the SVD and power iterations in order to extract the background information of some given video data. In this task, we use the opencv-python library to read the input video as a sequence of frames. And we develop a power-iteration based method to compute the singular value decomposition of our target matrix.

4.2 VIDEO MATRIX CONSTRUCTION

We utilize the opencv-python library to read the input video as a sequence of frames. To save storage, we convert each frame into a grayscale matrix, denoted by M_i, $M_i \in \mathbb{R}^{m\times n}$, $i \in \{1, 2, ..., s\}$. $\forall i$, put $m_i = vec(M_i)$ by stacking all the columns of M_i, $m_i \in \mathbb{R}^{mn}$. Then we construct our video matrix

$$A := [m_1 \quad m_2 \quad ... \quad m_s] \in \mathbb{R}^{mn\times s}. \tag{7}$$

To save memory storage, we use the following technique.

- **Frame Rescaling**
  Instead of using every frame in the video, we only pick some of the frames to form our video matrix. For example, in our first test video, we pick one every 30 frames, which can significantly save memory storage but we have to sacrifice some output performance. Take the video ”walking.mp4” as an example.
  - use every frame
  - 
  - one every 30 frames
  - 
  - one every 50 frames
  - 
  - one every 100 frames

- 
- one every 500 frames
- 
- one every 1000 frames
- 

use less frame could make our output image less ”pure”. As you can see, in this example, even the shape of a wheel could be seen if we use too few frame.

- **Resolution Resizing**
  Instead of storing the 640 $\times$ 360 images, we resize each selected frame to 240 $\times$ 135. However, using python package for svd, we do not encounter memory issue, since it may not need to construct such a big matrix.

- **Using Grayscale Image**

4.3 SINGULAR VALUE DECOMPOSITION OF VIDEO MATRIX

Our goal is to extract the largest singular value σ_1 and the corresponding left and right singular vector u_1 and v_1. Therefore, we do not need to compute all singular values and the

corresponding matrix U and V. We use the following technique.

Consider the video matrix $A \in \mathbb{R}^{mn \times s}, mn > s$.

Put $A_0 := \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix} \in \mathbb{R}^{(mn+s)\times(mn+s)}$. Now I claim that by applying power method on A_0, we can acquire σ_1, u_1, v_1.

$proof:$

Suppose (λ, x) is an eigen-pair of A_0, where$x \in \mathbb{R}^{s+mn}$ we have

$$A_0 x = \lambda x \tag{8}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, x_1 \in \mathbb{R}^s, x_2 \in \mathbb{R}^{mn} \tag{9}$$

$$\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A^T x_2 \\ A x_1 \end{bmatrix} = \begin{bmatrix} \lambda x_1 \\ \lambda x_2 \end{bmatrix} \tag{10}$$

$$\begin{cases} A^T x_2 = \lambda x_1 \\ A x_1 = \lambda x_2 \end{cases} \tag{11}$$

$$\begin{cases} A^T A x_1 = \lambda A^T x_2 = \lambda^2 x_1 \\ A A^T x_2 = \lambda A x_1 = \lambda^2 x_2 \end{cases} \tag{12}$$

from (11) we can see that λ is a singular value of A, x_2 is the corresponding left singular value and x_1 is the corresponding right singular value. Therefore, we only need one eigen-pair from A_0, which is exactly the one with largest modulus, naturally generated by power iteration applied on A_0.

4.4 ALGORITHM COMPARISON

please notice that we only compare the runtime of the singluar value decomposition part of the code, not the runtime of the entire code.

- **power method** 
- resolution rescaling: 240 $\times$ 135
- Runtime: 9.289251804351807s

- **svd package** 
- resolution rescaling: 240 $\times$ 135
- Runtime: 0.9243216514587402s
- **power method** 
- resolution rescaling: 240 $\times$ 135
- Runtime: 6.370991468429565s
- **svd package** 
- resolution rescaling: 240 $\times$ 135
- Runtime: 0.29717135429382324s

- **power method** 
- resolution rescaling: 240 $\times$ 135
- Runtime: 4.931594371795654s
- **svd package** 
- resolution rescaling: 240 $\times$ 135
- Runtime: 0.32863354682922363s
- **svd package** 
- resolution rescaling: 960 $\times$ 540
- Runtime: 4.630465507507324s

Since the auto package for singular value decomposition in python do not need to construct the large matrix A_0 in our algorithm, it can save a lot of space. Thus, using the package, we are able to increase the resolution of our output image without excceeding the limit of array strorage in numpy. In conclusion, the svd package in python is much more efficient than the power method we implement.

4.5 CONVERGENCE

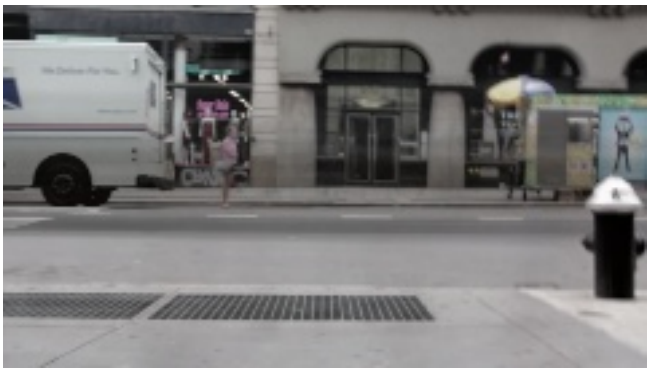
Here are the information of the number of steps of convergence and the corresponding tolerance of the first type video.

- tol: 0.01
- number of convergence step: 5
- tol: 0.0001
- number of convergence step: 6
- tol: 0.000001
- number of convergence step: 7

4.6 ADDITIONAL COLORED RECONSTRUCTION

Here we further develop the two methods on three channels and make some comparison

- 
- **4.6.1.1 power method**
- resolution rescaling: 240×135
- Runtime: 43.20660614967346s
- StepSize: Every 2 frames, ie. 666 out of 1331 frames
- 

- **4.6.1.2 svd package**
- resolution rescaling: 240 × 135
- Runtime: 71.58647894859314s
- StepSize: Every 2 frames, ie. 666 out of 1331 frames
- 
- **4.6.2.1 power method**
- resolution rescaling: 240 × 135
- Runtime: 51.98740911483765s
- StepSize: Every 1 frames, ie. 393 out of 393 frames
- 
- **4.6.2.2 svd package**
- resolution rescaling: 240 × 135
- Runtime: 38.48382902145386s
- StepSize: Every 1 frames, ie. 393 out of 393 frames

- 
- **4.6.3.1 power method**
- resolution rescaling: 240 × 135
- Runtime: 34.770559787750244s
- StepSize: Every 1 frames, ie. 415 out of 415 frames
- 
- **4.6.3.2 svd package**
- resolution rescaling: 240 × 135
- Runtime: 5.037815093994141s
- StepSize: Every 1 frames, ie. 415 out of 415 frames
- 
- **4.6.3.3 svd package**
- resolution rescaling: 960 × 540

- Runtime: 98.5620219707489s
- StepSize: Every 1 frames, ie. 415 out of 415 frames