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

DDA3005

NUMERICAL METHODS

INSTRUCTOR: DR. ANDRE MILZAREK

Course Project Singular Value Decomposition

Group Name: Macrohard

December 21, 2022

Group Members

ID	Name in Chinese	Name in English	Experiments Involved
120090549	温子雄	WEN Zixiong	1, 2, 3, 4, 5, 6
120090135	王子文	WANG Ziwen	1, 2, 3, 4, 5, 6
120090470	李鹏	LI Peng	1, 2, 3, 4, 5, 6
120090224	杨尚霖	YANG Shanglin	1, 2, 3, 4, 5, 6
120090771	邱纬纶	QIU Weilun	1, 2, 3, 4, 5, 6

Responsibilities and Contributions

This is a group project of the course DDA3005: Numerical Methods.

Contents

1 Summary of the Project

This is a course project of numerical methods which consists of three parts.

In the first part, we implement a two-phase procedure to perform singular value decomposition on a matrix A. In phase I, the matrix is first reduced to bidiagonal form via Golun-Kahan bidiagonalization. The resulting bidiagonal matrix B will be the input of the second phase. In phase II, different iterative procedure is applied to obtain the eigenvectors and eigenvalues of matrix B^TB. The SVD of B and A can then be constructed using these results.

In the second part, we apply SVD to image deblurring problem. The first step is to build two blurring kernels $A_l \in \mathbb{R}^{n\times n}$ and $A_r \in \mathbb{R}^{n\times n}$, where n is the size of the image. Here, we adopt two different models in the application, which will be discussed in the third section. The blurry image can be consturcted using the two kernels. In the second step, the truncation technique is used to generate the pseudoinverse A_l^+ and A_r^+. The blurry images are deblurred using the trancated reconstruction of persudoinverses. We calculate peak-signal-to-noise-ratio (PSNR) to measure the quality of reconstruction and compare the runtimes of SVDs using the two iterative procedures mentioned in the first part.

2 Singular Value Decomposition

2.1 About Part 1

In the first part of the project, we implement the **singular value decomposition (SVD)** of a matrix using a **two-phase procedure**. An SVD of a matrix $A \in \mathbb{R}^{m\times n}$ factorizes the matrix in the following form.

$$A = U\Sigma V^T \tag{2.1}$$

The matrix U is generated by stacking eigenvectors of the matrix $AA^T \in \mathbb{R}^{m\times m}$. The matrix V follows a similar custom, which is formed by stacking eigenvectors of the matrix $A^TA \in \mathbb{R}^{n\times n}$. The matrix $\Sigma \in \mathbb{R}^{m\times n}$ contains the information of singular values $\sigma_i,\ i = 1, 2, \cdots, \min\{n, m\}$ at its main diagonal.

The two phases of the procedure is described as following.

- Phase I: Conduct the **Golub-Kahan bidiagonalization** to reduce the matrix A to bidiagonal form. The resulting matrix is denoted as B.

- Phase II-A: Perform **QR iteration** with **Wilkinson shift** and **deflation** on matrix B^TB. The output of this phase will be the eigenvectors and the eigenvalues of B^TB.

With the above outputs, the SVD of matrix B can be constructed. The SVD of the original matrix A can be constructed based on the SVD of B.

Besides, an alternative method of the QR iteration is adopted in phase II-B, which follows the following scheme.

- Phase II-B: Replace the QR iteration with the following iteration.

$$\begin{aligned} &\text{Initilize } X^0 = B \text{ and for } k = 0, 1, \cdots \text{ do:} \\ &Q_kR_k = (X^k)^T, \quad L_kL_k^T = R_kR_k^T, \quad X^{k+1} = L_k^T \end{aligned} \tag{2.2}$$

This iterative procedure is in fact equivalent to the original QR iteration, which will be varified in the following sections.

2.2 Algorithmic Component I - Golub-Kahan bidiagonalization

Following the two-phase approach scheme, the first step is to bidiagonalize the matrix in an iterative way. For the bidiagonal matrix constructed in this project, the main diagonal and the diagonal above consist of nonzero entries. Therefore, in each iteration we apply two householder reflections on the left and right. The left one introduces zeros to entries below the diagonal while the right one introduces zeros to the right of the superdiagonal.

$$\begin{aligned} &\text{for } k = 0, 1, \cdots, \text{ let } a_k \text{ be the } (k+1)\text{-th column of the matrix and do:} \\ &v_k = a_k \pm \|a_k\|e_k, \quad Hv_k = I_k - 2\frac{v_kv_k^T}{\|v\|_2^2}, \quad A_{k+1} \to \begin{bmatrix} I_k & \mathbf{0} \\ \mathbf{0} & Hv_k \end{bmatrix} \cdot A_k \end{aligned} \tag{2.3}$$

The above scheme shows the procedure of householder transformation on columns. The procedure of row reducing is similar to this scheme except that each row vector b_k does not need to include the element in the main diagonal. The general procedure of bidiagonalization is shown as following.

$$\begin{aligned} &\text{for } k = 0, 1, \cdots, \text{ do:} \\ &\text{build } U_k^T \text{ using scheme (1.3) on column k} \\ &\text{build } V_k \text{ using scheme (1.3) on row k} \\ &A_{k+1} \to U_k^TA_kV_k \end{aligned} \tag{2.4}$$

2.3 Algorithmic Component II - QR iteration with Wilkinson Shift and Deflation

The general procedure of the entire algorithm is described by the pesudocode posted in Appendix-1.1. The algorithm takes a square matrix $A \in \mathbb{R}^{n\times n}$ as input and output a list of eigenvalues $\lambda_i,\ i = 1, \cdots, n$ and a matrix Q whose columns are the eigenvectors of the correponding eigenvalues in the list. The Wilkinson shift is computed by calculating the eigenvectors of the 2×2 matrix at the lower right corner of the matrix. The shift value will be selected to be the eigenvalues which is closer to the value in the entry $A(r, r),\ r = 1, \cdots, n$. As for deflation, the algorithm keeps track of the norm of the vector $A(1 : r - 1, r)$ and once it is smaller than the preset tolerence, deflation is conducted. In actual implementation, the tolerence is set as $\text{tol} = 1\text{e}{-11}$.

2.4 Algorithmic Component III - alternative iterative procedure of QR iteration

In this part, the iterative procedure described in (1.2) is implemented. This procedure actually coincides with the QR iteration with zero shift. The equivalence will be shown as following.

For the k-th iteration of the QR algorithm, we have that

$$\begin{aligned} Q_k R_k &= X^{k-1} \\ X^k &= R_k Q_k. \end{aligned} \tag{2.5}$$

Therefore, $R_k = Q_k^T X^{k-1}$ and $X^k = Q_k^T X^{k-1} Q_k$ are satisfied. If this result is applied to all the previous iterations, we have

$$X^k = Q_k^T Q_{k-1}^T \cdots Q_1^T X^0 Q_1 \cdots Q_{k-1} Q_k. \tag{2.6}$$

Therefore, when $B^T B$ is applied to QR iteration, we have that

$$X^k = Q_k^T Q_{k-1}^T \cdots Q_1^T B^T B Q_1 \cdots Q_{k-1} Q_k \tag{2.7}$$

Similarly, in the k-th iteration of our alternative method (2.2), we have $R_k = Q_k^T (X^k)^T$. Since $X^{k+1} = L_k^T$ and $L_k L_k^T = R_k R_k^T$ is satisfied, we have that

$$(X^{k+1})^T X^{k+1} = Q_k^T (X^k)^T X^k Q^k \tag{2.8}$$

Apply the above result to all the previous iterations, and set $X^0 = B$, we have

$$(X^{k+1})^T X^{k+1} = Q_k^T Q_{k-1}^T \cdots Q_1^T Q_0^T B^T B Q_0 Q_1 \cdots Q_{k-1} Q_k. \tag{2.9}$$

It turns out that $(X^{k+1})^T X^{k+1}$ in equation (2.9) is equivalent to X^k in equation (2.7). The difference is that the diagonal elements of the output X^k in QR iteration converge to eigenvalues of B^TB, while in the alternative method, they converge to singular values directly.

2.5 Main Results and Observation

An example of the output of the program can be seen in Appendix-1.2. The number of iterations of the alternative iteration is set as 1000. From the results, both methods are able to give reliable results.

3 Deblurring Revisited

3.1 About Part II

For this part of the project, the SVD program implemented in part 1 is utilized to finish deblurring tasks. In this project, deblurring problems with the following form are considered.

$$B = A_l X A_r \tag{3.1}$$

In above equation, $X \in \mathbb{R}^{n\times n}$ represents the matrix of original image. The resulting blurry image will be $B \in \mathbb{R}^{n\times n}$. The matrix $A_l \in \mathbb{R}^{n\times n}$ and $A_r \in \mathbb{R}^{n\times n}$ are the left and right blurring kernels. To reconstruct the original image from a given blurry image B and the two blurring kernels, we need to compute the pesudoinverses A_l^+ and A_r^+ and the recovered image can be obtained by $X = A_l^+ B A_r^+$.

The first step of this part is to construct the blurring kernels. Two different models are used to blur the images.

- Model I: We set $A_l = A_r = T^k$ where $k = 40$ and matrix T has the following form.

$$T = \begin{bmatrix} \frac{2+\delta}{4+\delta} & \frac{1}{4+\delta} & & & \\ \frac{1}{4+\delta} & \frac{2+\delta}{4+\delta} & \frac{1}{4+\delta} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \frac{1}{4+\delta} \\ & & & \frac{1}{4+\delta} & \frac{2+\delta}{4+\delta} \end{bmatrix} \in \mathbb{R}^{n\times n},\ \delta = 0.1 \tag{3.2}$$

- Model II: The matrix A_l and A_r are both symmetric and have the following form.

$$A = \begin{bmatrix} a_n & a_{n-1} & & a_2 & a_1 \\ a_{n-1} & a_n & a_{n-1} & & a_2 \\ & \ddots & \ddots & \ddots & \\ a_2 & & \ddots & \ddots & a_{n-1} \\ a_1 & a_2 & & a_{n-1} & a_n \end{bmatrix}, \quad \Sigma_{i=1}^{n} a_i = 1, \quad a_i \geq 0 \tag{3.3}$$

  In this case, A_l and A_r are different symmetric matrices and the blurring kernel computes the weighted average of the pixels to generate the blurry image.

To compute the inverse (or pesudoinverse) of A_l and A_r, the trancated SVD technique is applied. According to SVD of a matrix A with rank r, we have that $A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T$. Therefore, the matrix A^+ can be computed as $A^+ = V\Sigma^{-1}U^T = \sum_{i=1}^{r} \frac{1}{\sigma_i} v_i u_i^T$. With additional parameters l_{tranc} and r_{tranc}, the two matrices can be constructed as

$$A^+_{l,\text{trunc}} = \sum_{i=1}^{l_{\text{tranc}}} \frac{v_i^l (u_i^l)^T}{\sigma_i}, \; A^+_{r,\text{trunc}} = \sum_{i=1}^{r_{\text{tranc}}} \frac{v_i^r (u_i^r)^T}{\sigma_i} \tag{3.4}$$

3.2 Main Results and Observation

3.2.1 Quality of Reconstruction in Model I

The quality of the reconstruction is related to the choice of the parameter l_{tranc} and r_{tranc}.

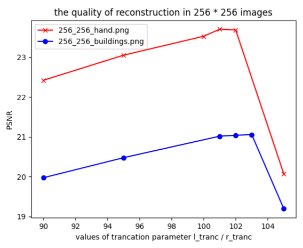


Figure 1: 256 × 256 image

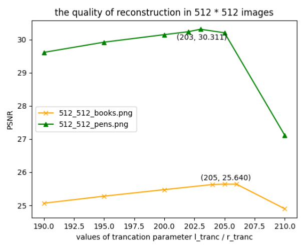


Figure 2: 512 × 512 image

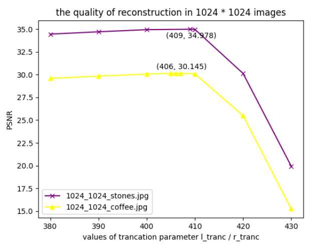


Figure 3: 1024 × 1024 image

The above three figures show the results obtained by applying 6 images to model I with various sizes. An observation of the relation between peak-signal-to-noise ratio (PSNR) and trancation parameters l_{trunc} and r_{trunc} is that the PSNR increases as the trancation parameters increase when $l_{\text{trunc}} = r_{\text{trunc}} < l^*$. When $l_{\text{trunc}} = r_{\text{trunc}} = l^*$, the reconstructed image will have the largest PSNR. When the trancation parameters are larger than the optimal value l^*, the PSNR of the reconstruction sharply decreases. Another observation is that the value of the thereshold l^* depends on the size of the image. From Figure 1, the trancation value which gives the best quality of reconstruction is around 102 (101 for 256_256_hand.png and 103 for 256_256_buildings.png). As for 512×512 images, the optimal value of l_{trunc} and r_{trunc} is around 203. For 1024×1024 images, this value will change to around 406.

3.2.2 Quality of Reconstruction in Model II

The experiment on model II presents a result that is different from model I.

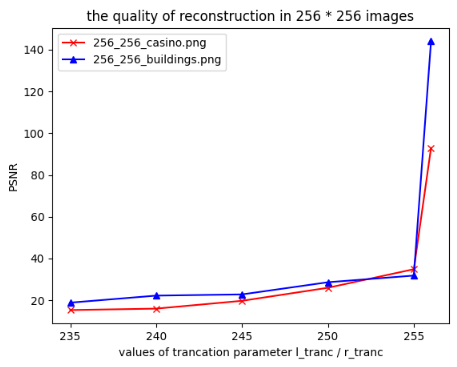


Figure 4: 256×256 image

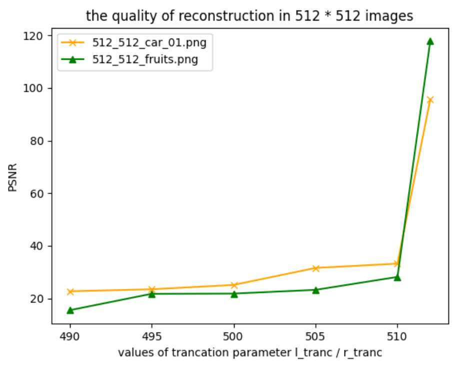


Figure 5: 512×512 image

In this case, the PSNR increases as the value of the trancation parameter increases. In addition, the PSNR increases sharply when the last singular value and singular vectors are applied to construct pseudoinverse. The corresponding data and the effect of deblurring in this model are shown in Appendix-2.2.

4 Appendix

4.1 Appendix-1.1: procedure of QR iteration using Wilkinson Shift and Deflation

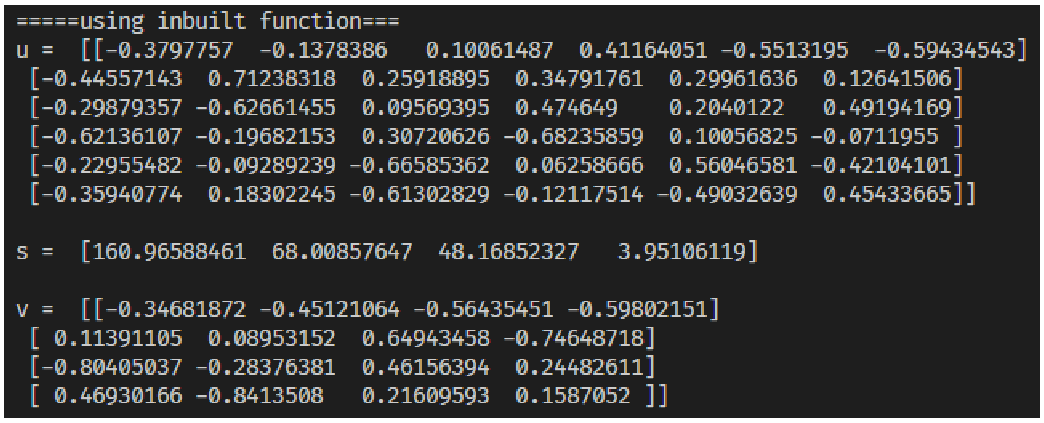
```
input: $A \in \mathbb{R}^{n \times n}$
set $Q = Q_{\text{tmp}} = I_n \in \mathbb{R}^{n \times n}$
for $r = n$ to 2 do:
  $k = 0, \quad X^0 = A(1:r, 1:r)$
  while True:
    $k = k + 1$
    $\sigma_{k-1} = \text{Wilkinson\_shift}(X^{k-1})$
    $Q_k R_k = X^{k-1} - \sigma_{k-1} I$
    $X^k = R_k Q_k + \sigma_{k-1} I$
    $Q_{\text{tmp}} = Q_{\text{tmp}} \cdot Q_k$
    if $X^k(1:r-1, r) < tol$:
      $\lambda_r = X^k(r, r)$
      $Q = Q \cdot \begin{bmatrix} Q_{\text{tmp}} & \mathbf{0} \\ \mathbf{0} & I_{n-r} \end{bmatrix}$
      break
$\lambda_1 = X^k(1, 1)$
end
output: $\lambda_i,\ i = 1, \cdots, n$; $Q \in \mathbb{R}^{n \times n}$
```

4.2 Appendix-1.2: example of the output in part 1

In this example, the following matrix A is applied to the program.

$$A = \begin{bmatrix} 17 & 21 & 31 & 45 \\ 21 & 32 & 78 & 10 \\ 9 & 15 & 2 & 62 \\ 20 & 42 & 54 & 73 \\ 38 & 25 & 2 & 19 \\ 45 & 36 & 27 & 18 \end{bmatrix}$$

To varify whether the QR iteration or the alternative iterative procedure give a precise result, the python inbuilt function numpy.linalg.svd is used to compare with the output.

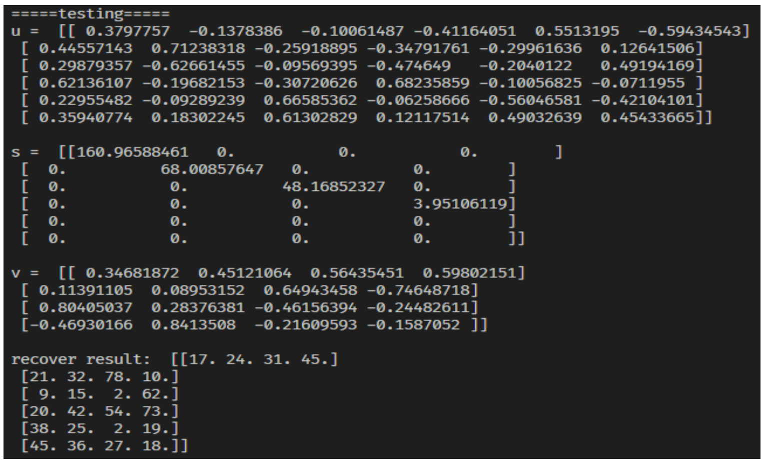


```
=====using inbuilt function===
u =  [[-0.3797757  -0.1378386   0.10061487  0.41164051 -0.5513195  -0.59434543]
 [-0.44557143  0.71238318  0.25918895  0.34791761  0.29961636  0.12641506]
 [-0.29879357 -0.62661455  0.09569395  0.474649    0.2040122   0.49194169]
 [-0.62136107 -0.19682153  0.30720626 -0.68235859  0.10056825 -0.0711955 ]
 [-0.22955482 -0.09289239 -0.66585362  0.06258666  0.56046581 -0.42104101]
 [-0.35940774  0.18302245 -0.61302829 -0.12117514 -0.49032639  0.45433665]]

s =  [160.96588461  68.00857647  48.16852327   3.95106119]

v =  [[-0.34681872 -0.45121064 -0.56435451 -0.59802151]
 [ 0.11391105  0.08953152  0.64943458 -0.74648718]
 [-0.80405037 -0.28376381  0.46156394  0.24482611]
 [ 0.46930166 -0.8413508   0.21609593  0.1587052 ]]
```

Figure 6: the SVD of A using python inbuilt function



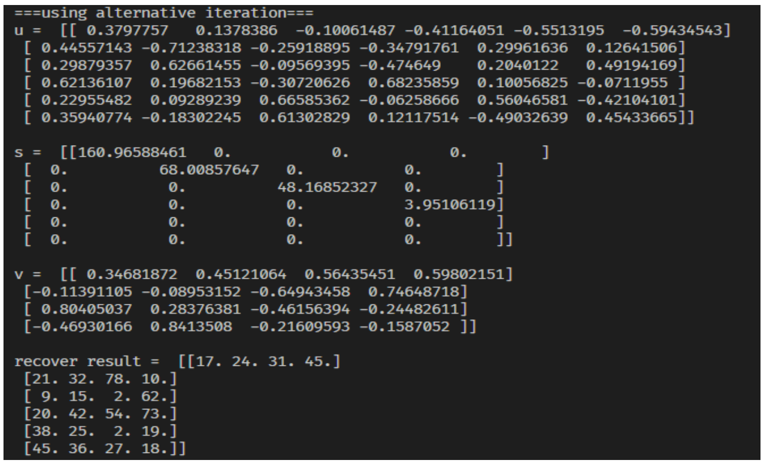
```
=====testing=====
u =  [[ 0.3797757  -0.1378386  -0.10061487 -0.41164051  0.5513195  -0.59434543]
 [ 0.44557143  0.71238318 -0.25918895 -0.34791761 -0.29961636  0.12641506]
 [ 0.29879357 -0.62661455 -0.09569395 -0.474649   -0.2040122   0.49194169]
 [ 0.62136107 -0.19682153 -0.30720626  0.68235859 -0.10056825 -0.0711955 ]
 [ 0.22955482 -0.09289239  0.66585362 -0.06258666 -0.56046581 -0.42104101]
 [ 0.35940774  0.18302245  0.61302829  0.12117514  0.49032639  0.45433665]]

s =  [[160.96588461   0.            0.            0.        ]
 [  0.          68.00857647   0.            0.        ]
 [  0.           0.          48.16852327   0.        ]
 [  0.           0.           0.           3.95106119]
 [  0.           0.           0.           0.        ]
 [  0.           0.           0.           0.        ]]

v =  [[ 0.34681872  0.45121064  0.56435451  0.59802151]
 [ 0.11391105  0.08953152  0.64943458 -0.74648718]
 [ 0.80405037  0.28376381 -0.46156394 -0.24482611]
 [-0.46930166  0.8413508  -0.21609593 -0.1587052 ]]

recover result:  [[17. 24. 31. 45.]
 [21. 32. 78. 10.]
 [ 9. 15.  2. 62.]
 [20. 42. 54. 73.]
 [38. 25.  2. 19.]
 [45. 36. 27. 18.]]
```

Figure 7: the SVD of A using QR iteration with wilkinson shift and deflation



```
===using alternative iteration===
u =  [[ 0.3797757   0.1378386  -0.10061487 -0.41164051 -0.5513195  -0.59434543]
 [ 0.44557143 -0.71238318 -0.25918895 -0.34791761  0.29961636  0.12641506]
 [ 0.29879357  0.62661455 -0.09569395 -0.474649    0.2040122   0.49194169]
 [ 0.62136107  0.19682153 -0.30720626  0.68235859  0.10056825 -0.0711955 ]
 [ 0.22955482  0.09289239  0.66585362 -0.06258666  0.56046581 -0.42104101]
 [ 0.35940774 -0.18302245  0.61302829  0.12117514 -0.49032639  0.45433665]]

s =  [[160.96588461   0.           0.           0.        ]
 [  0.          68.00857647   0.           0.        ]
 [  0.           0.          48.16852327   0.        ]
 [  0.           0.           0.           3.95106119]
 [  0.           0.           0.           0.        ]
 [  0.           0.           0.           0.        ]]

v =  [[ 0.34681872  0.45121064  0.56435451  0.59802151]
 [-0.11391105 -0.08953152 -0.64943458  0.74648718]
 [ 0.80405037  0.28376381 -0.46156394 -0.24482611]
 [-0.46930166  0.8413508  -0.21609593 -0.1587052 ]]

recover result =  [[17. 24. 31. 45.]
 [21. 32. 78. 10.]
 [ 9. 15.  2. 62.]
 [20. 42. 54. 73.]
 [38. 25.  2. 19.]
 [45. 36. 27. 18.]]
```

Figure 8: the SVD of A using alternative iterative procedure

4.3 Appendix-2.1: examples and data obtained when measuring the quality of reconstruction in model I

* 256×256 images

image	trancation number	PSNR
256_256_hand.png	90	22.419
	95	23.049
	100	23.525
	101	23.701
	102	23.683
	105	20.068
256_256_buildings.png	90	19.972
	95	20.475
	101	21.016
	102	21.038
	103	21.056
	105	19.193

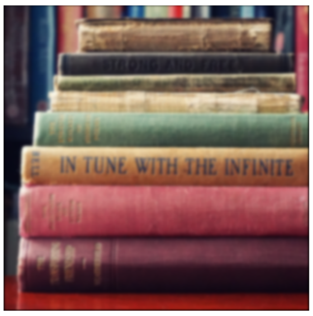
* 512×512 images

image	**trancation number**	**PSNR**
512__512__books.png	190	25.065
	195	25.278
	200	25.475
	204	25.629
	205	25.640
	206	25.639
	210	24.897
512__512__pens.png	190	29.613
	195	29.921
	200	30.148
	204	30.237
	205	30.311
	206	30.204
	210	27.112

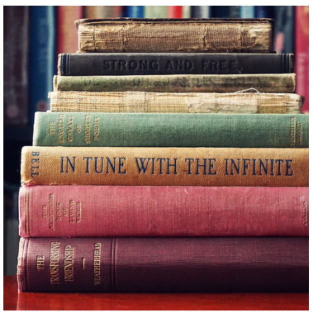
* 1024×1024 images

image	**trancation number**	**PSNR**
1024__1024__stones.png	380	34.443
	390	34.698
	400	34.932
	409	34.978
	410	34.964
	420	30.124
	430	19.935
1024__1024__coffee.png	380	29.576
	390	29.827
	400	30.054
	405	30.131
	406	30.145
	407	30.137
	410	30.093
	420	25.506
	430	15.266

* examples of blurring and deblurring an image using model I

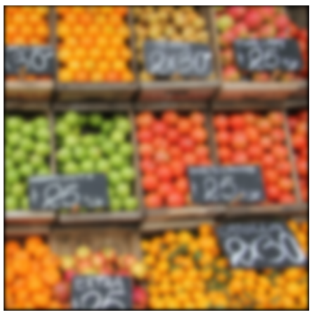


(a) blurry image of 1024_1024_books.png

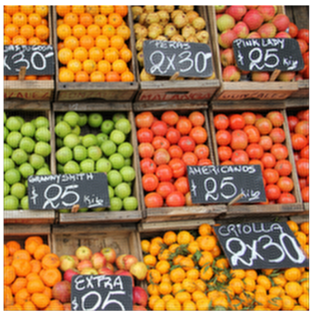


(b) reconstruct image of 1024_1024_books.png

Figure 9: blurry and reconstructed images of 1024_1024_books.png with $l_{\text{trunc}} = r_{\text{trunc}} = 406$



(a) blurry image of 512_512_fruits.png



(b) reconstruct image of 512_512_fruits.png

Figure 10: blurry and reconstructed images of 512_512_fruits.png with $l_{\text{trunc}} = r_{\text{trunc}} = 203$