Software Assignment - Image Compression Using Truncated SVD

EE25Btech11063 - Y. Vejith

November 8, 2025

Introduction

A greyscale image can be represented as a matrix A $\in \mathbb{R}^{m\times n}$, where each entry of the matrix a_{ij} corresponds to the pixel intensity (0 = black, 255 = white).

The **Singular Value Decomposition (SVD)** of the matrix A can be expressed as:

$$A = U\Sigma V^\top \tag{1}$$

By Taking the top k eigen values we have to reconstruct the matrix A as

$$A_k = U_k\Sigma_k V_k^\top \tag{2}$$

After reconstruction of matrix A we should compare it′s various properties like quality, etc

Singular Value Decomposition by Strang

Eigen value decomposition can be performed only on symmetric matrices.The decomposition which can be implemented on any matrix of arbitrary size is singular value decomposition

let A $\in \mathbb{R}^{m\times n}$ be a matrix of any size the **SVD** of A can be shown as

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

where r is the rank of the matrix A

$$U \in \mathbb{R}^{m \times r} \text{is a matrix having it's columns as orthonormal eigen vectors of } \mathrm{AA}^\top \tag{4}$$

$$V \in \mathbb{R}^{n \times r} \text{is a matrix having it's columns as orthonormal eigen vectors of } \mathrm{A}^\top \mathrm{A} \tag{5}$$

$$\Sigma^2 \in \mathbb{R}^{r \times r} \text{is a diagonal matrix having eigen values of } \mathrm{A}^\top \mathrm{A} \text{ in decreasing order} \tag{6}$$

$$\Sigma^2 = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r^2 \end{bmatrix} \tag{7}$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ are the nonzero singular values of the matrix A.

Image to intensity matrix

The images in .jpg and in .png format can be coverted into intensity matrix by using a library in c language called **stb_image.h**
The macro STB_IMAGE_IMPLEMENTATION must be included before the header
This tells the compiler to include the function implementations (actual code) from the header file, allowing functions like stbi_load() and stbi_image_free()

1 Code

```
int width, height, channels;
unsigned char *img =stbi_load(filename, &width, &height, &channels, 1);
```

value of the channel is 1 for a greyscale image
stbi_load() returns an integer pointer to width and height of the image

```
fprintf(fp, "%d %d\n", height, width);
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            int p = img[i * width + j];
            fprintf(fp, "%3d ", pixel);
        }
        fprintf(fp, "\n");
    }
```

writes the height and width of the image in the 1st line of the file matrix.txt and stores the intensity values in the file
Now the pixel intensities are stored in a file called matrix.txt

Truncated SVD

There are many methods to perform truncated svd but the method which has less computational cost and more efficiency is **Randomized SVD using Power Iterations**

Randomized SVD using Power Iterations

Mathematical Background
let A $\in \mathbb{R}^{m\times n}$ be a matrix.Then,

$$A = U\Sigma V^{\top} \tag{8}$$

Computing full SVD for A is of order $O(mn^2)$ which is very expensive for large matrices like 512×512 and 1024×1024.so we use **randomization** to approximate the column space of A
Generate a Random matrix Ω

$$\Omega \in \mathbb{R}^{n\times k} \tag{9}$$
$$\Omega_{ij} \sim N(0,1) \tag{10}$$

(10) means the entries of Ω are choosen such that average of all the entries is 0 and the variance is 1

$$Y = A\Omega \tag{11}$$

where Y $\in \mathbb{R}^{m\times k}$ which interprets that each column of Y is a linear combination of colums of A
$\therefore$ Y contain almost the same subspace information as the top k singular vectors of A
Power Iterations
If singular values of A decay slowly then (11) may not be accurate so to make it more efficient we use power iterations

$$Y = (AA^{\top})^{\mathrm{q}} A\Omega \tag{12}$$

where q is the number of power iterations typically q = 1 or 2 but q =2 handles very large matrices also very efficiently. q larger than 2 also gives nearly same accuracy as q = 2 but it′s computationally expensive.so let′s stick to q = 2

$$\begin{aligned} A &= U\Sigma V^\top & (13)\\ AA^\top &= U\Sigma^2 U^\top & (14)\\ Y &= (AA^\top)^{\mathrm{q}} A\Omega & (15)\\ \implies Y &= U\Sigma^{2\mathrm{q}+1} V^\top \Omega & (16) \end{aligned}$$

After this operation each singular value of the matrix is raised by 2q + 1 times which makes Y to capture the dominant k values efficiently

QR decomposition

Now we have to find the orthogonal basis of Y and make the columns to unit norm for that we can use QR decomposition based on Gram-schmidt algorithm

let

$$a_1, a_2 \ldots a_{\mathrm{k}} \text{ be the columns of the matrix Y} \quad (17)$$

$$q_1, q_2 \ldots q_{\mathrm{k}} \text{ be the orthonormal columns of the matrix Q} \quad (18)$$

$$\begin{aligned} q_1 &= \frac{a_1}{||a_1||} & (19)\\ \tilde{q_2} &= a_2 - (a_2^\top q_1)\, q_1 & (20)\\ q_2 &= \frac{\tilde{q_2}}{||\tilde{q_2}||} & (21) \end{aligned}$$

we can generalize for q_p for $p \in [1, \mathrm{k}]$as

$$\begin{aligned} \tilde{q}_p &= a_p - \sum_{j=1}^{p-1} \left(a_p^\top \cdot q_j\right) q_j & (22)\\ q_p &= \frac{\tilde{q}_p}{\left\|\tilde{q}_p\right\|} & (23) \end{aligned}$$

Finally

$$Q = [\, q_1 \; q_2 \; \cdots \; q_k \,] \text{ and } Q \in \mathbb{R}^{m\times k} \quad (24)$$

Q is the matrix whose columns are orthonormal and are basis of Y

Q approximates the span of top K singular values of A

$$A \approx QQ^\top A \quad (25)$$

Projection
let′s project A into this k-dimensional subspace(Q)

$$B = Q^\top A \tag{26}$$

B $\in \mathbb{R}^{k\times n}$
B is a smaller matrix than A which retains the structure of A so it is easy to perform SVD on B than A
SVD

$$B = \tilde{U}\Sigma V^\top \tag{27}$$

where

$$\tilde{U} \in \mathbb{R}^{k\times k}\text{is a matrix having it's columns as the top k eigen vectors of BB}^\top \tag{28}$$
$$V \in \mathbb{R}^{n\times k}\text{is a matrix having it's columns as the top k eigen vectors of A}^\top\text{A} \tag{29}$$
$$\Sigma \in \mathbb{R}^{k\times k}\text{is a matrix having the top k singular values of matrix A} \tag{30}$$

Relation between U and $\tilde{U}$

$$B = Q^\top A \tag{31}$$
$$B^\top = A^\top Q \tag{32}$$
$$\implies BB^\top = (Q^\top A)(A^\top Q) \tag{33}$$
$$\implies BB^\top = Q^\top AA^\top Q \tag{34}$$

From (27)

$$BB^\top = \tilde{U}\Sigma\tilde{U}^\top \tag{35}$$
$$\implies Q^\top AA^\top Q = \tilde{U}\Sigma^2\tilde{U}^\top \tag{36}$$
$$\implies Q^\top AA^\top Q\tilde{U} = \tilde{U}\Sigma^2 \tag{37}$$

Multiply Q on left of (37)

$$QQ^\top AA^\top Q\tilde{U} = Q\tilde{U}\Sigma^2 \tag{38}$$

From (25) (38) can be written as

$$AA^\top Q\tilde{U} = Q\tilde{U}\Sigma^2 \tag{39}$$
$$AA^\top = Q\tilde{U}\Sigma^2\left(Q\tilde{U}\right)^{-1} \tag{40}$$

AS Q is orthogonal and $\tilde{U}$ is also orthogonal Q$\tilde{U}$ is also orthogonal

$$\left(Q\tilde{U}\right)^{-1} = \left(Q\tilde{U}\right)^{\top} \tag{41}$$

using (41) (40) can be written as

$$AA^{\top} = Q\tilde{U}\Sigma^{2}\left(Q\tilde{U}\right)^{\top} \tag{42}$$

As Σ^2 contains the squares of top K singular values the matrix Q$\tilde{U}$ should contain the eigen vectors corresponding to the top k values so we can conclude that

$$U = Q\tilde{U} \tag{43}$$

let A$_k$ be the matrix reconstructed using the top k singular values

$$\implies A_k = U\Sigma V^{\top} \tag{44}$$

From (43)

$$A_k = Q\tilde{U}\Sigma V^{\top} \tag{45}$$

$$\implies \mathbf{A}_k \mathbf{= QB} \tag{46}$$

$$Q \in \mathbb{R}^{m\times k} \text{ and } B \in \mathbb{R}^{k\times n} \tag{47}$$

2 Codes

2.1 Create a matrix

```
    double **creatematrix(int m,int n){
 double **mat = (double **)malloc(m*sizeof(double *));
 for(int i=0;i<m;i++){
  mat[i] = (double *)malloc(n*sizeof(double)); }
  return mat;
  }
```

2.2 Random matrix

```
    double **randommatrix(int m,int n){
 double **rand = creatematrix(m,n);
  for(int i=0;i<m;i++){
   for(int j=0;j<n;j++){
     rand[i][j] = ((double)rand()/ RAND_MAX);
     }}
     return rand;
}
```

2.3 QR Decomposition

```
void Gram-schmidt(double **A,int m,int n){
 for(int i=0;i<n;i++){
   for(int j=0;j<i;j++){
     double res=0;
    for (int k = 0; k < m; k++)
              res += A[k][i] * A[k][j];
          for (int k = 0; k < m; k++)
              A[k][i] -= res * A[k][j];
     }
     double norm =0;
     for(int p=0;p<m;p++){
       norm += A[p][i] *A[p][i];
       }
       norm =sqrt(norm);
   for (int k = 0; k < m; k++){
              A[k][i] = A[k][i]/norm;
   }
   }}
```

Error Analysis

Frobenius norm
let A $\in \mathbb{R}^{m \times n}$ have SVD as

$$A = U\Sigma V^{\top} = \sum_{i=1}^{r} \sigma_i U_i V_i^T \tag{48}$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ are the nonzero singular values of the matrix A. The rank-*k* truncated reconstruction of matrix A using the top *k* singular values is given by

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T \tag{49}$$

The Frobenius norm of the error $||A - A_k||$ is

$$||A - A_k||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (A - A_k)_{ij}^2} \tag{50}$$

3 Code

```
    double frobeniusnorm(double **A, double **B, int m, int n) {
    double res = 0.0;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++) {
            double d = A[i][j] - B[i][j];
            res += d * d;
        }
    return sqrt(res);
}
```

Intensity matrix to Image

Now we have to construct the image using the matrix which is reconstructed using the top *k* singular values. Based on input it will decide whether it is a .png or .jpg file based on that we can use the respective functions in stb_image_write.h

4 code

```
    char *ext = strrchr(output, '.');
    if (ext && strcmp(ext, ".png") == 0) {
        if (stbi_write_png(output, rwidth, rheight, 1, rimg, rwidth))
            printf("Image saved as %s\n", output);
        else
            printf("Failed to save the image.\n");
    }
    else if (ext && strcmp(ext, ".jpg") == 0) {
        if (stbi_write_jpg(output, rwidth, rheight, 1, rimg, 100))
            printf("Image saved as %s\n", output);
        else
            printf("Failed to save the image.\n");
    }
    else {
        printf("Unsupported file extension. \n");
    }
```

Implementation

5 globe.jpg

original image

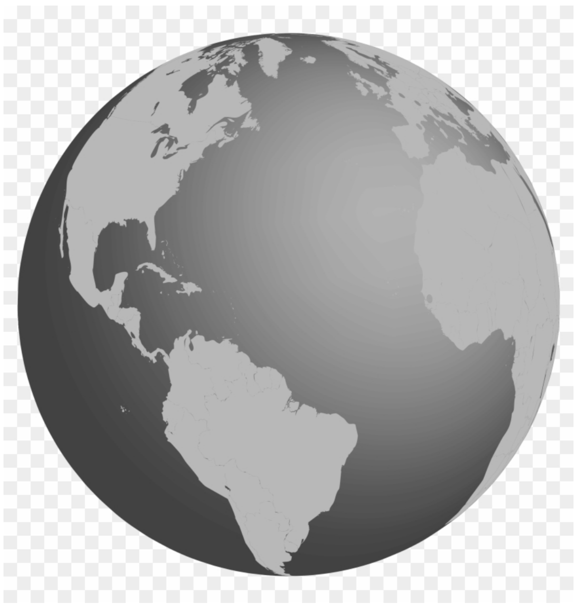

Figure 0: Given image

Name	Value of k	Frobenius norm
globe_10	10	15748.232558
globe_50	50	6813.587061
globe_200	200	6712.990925
globe_500	500	5965.878649
globe_830	830	5507.882616

Table 0: Relation between k and frobenius norm

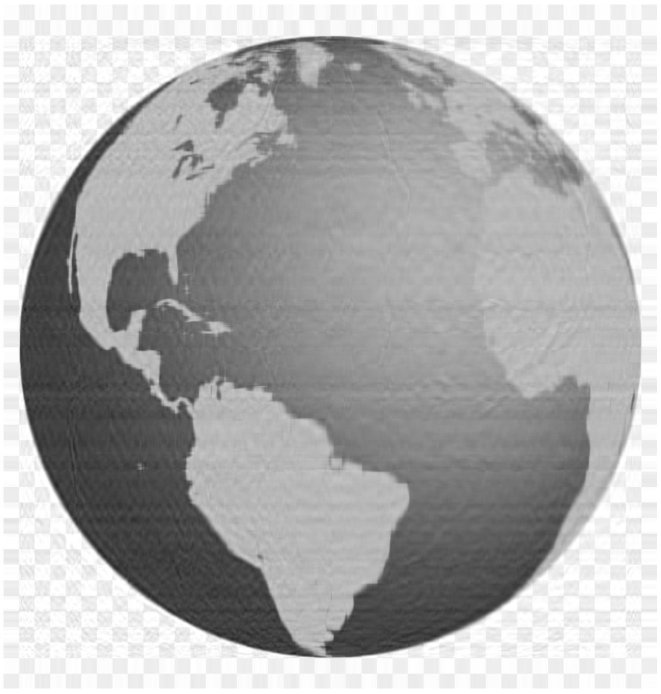

Figure 0: Image of globe at k= 830

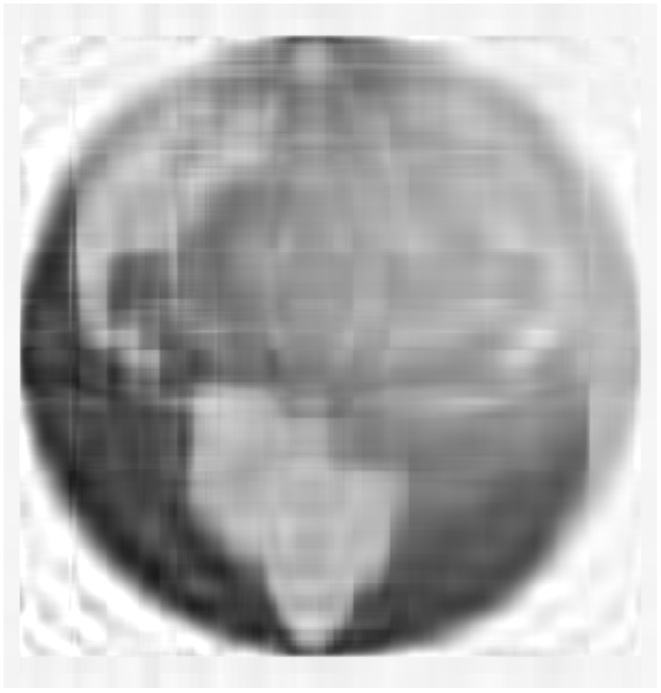

Figure 0: Image of globe at k= 10

6 einstein.jpg

original image

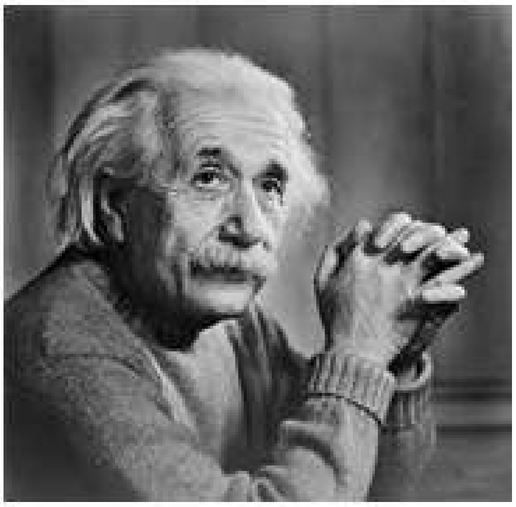

Figure 0: Given image

Name	**Value of k**	**Frobenius norm**
einstein10	10	4602.906631
einstein50	50	1096.796928
einstein100	100	625.877457
einstein150	150	616.945708
einstein180	180	606.388071

Table 0: Relation between k and frobenius norm

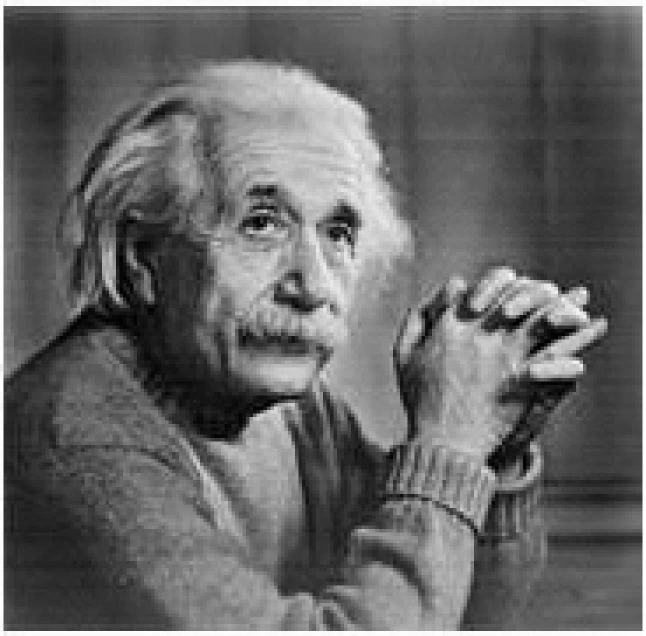

Figure 0: Image of einstein at k= 180

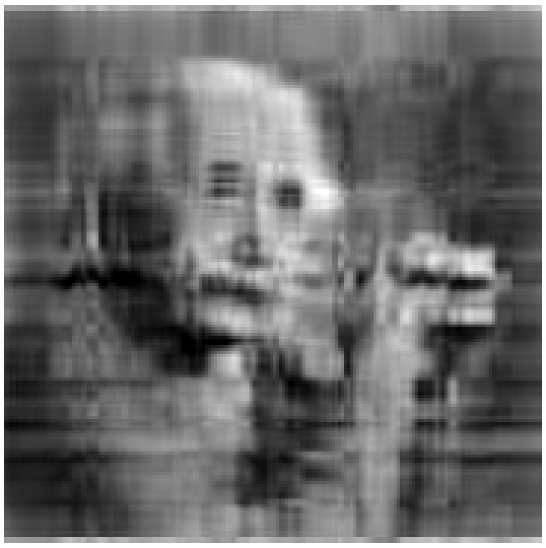

Figure 0: Image of einstein at k= 10

7 greyscale.png

original image

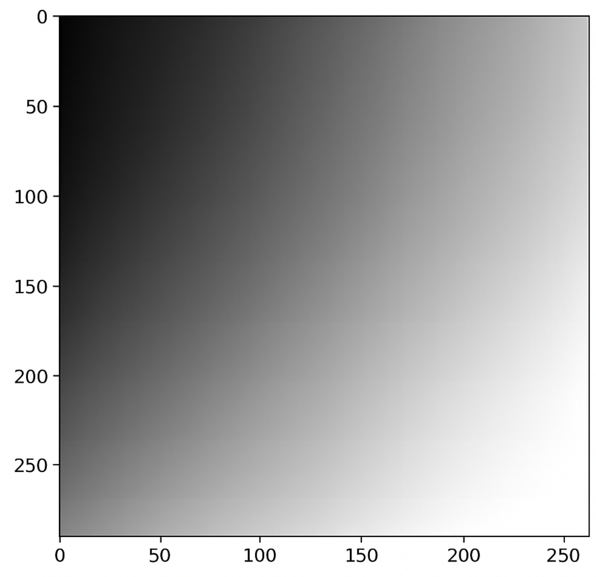

Figure 0: Given greyscale image

Name	Value of k	Frobenius norm
greyscale10	10	8548.727680
greyscale100	100	6069.933135
greyscale500	200	4656.995356
greyscale1000	500	4503.574645
greyscale1024	1000	4414.255277

Table 0: Relation between k and frobenius norm

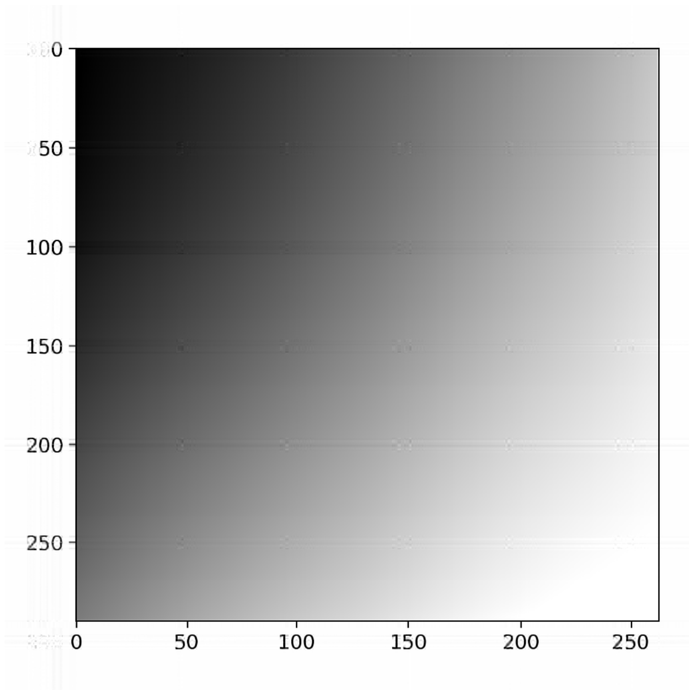

Figure 0: Image of globe at k= 1024 i.e max

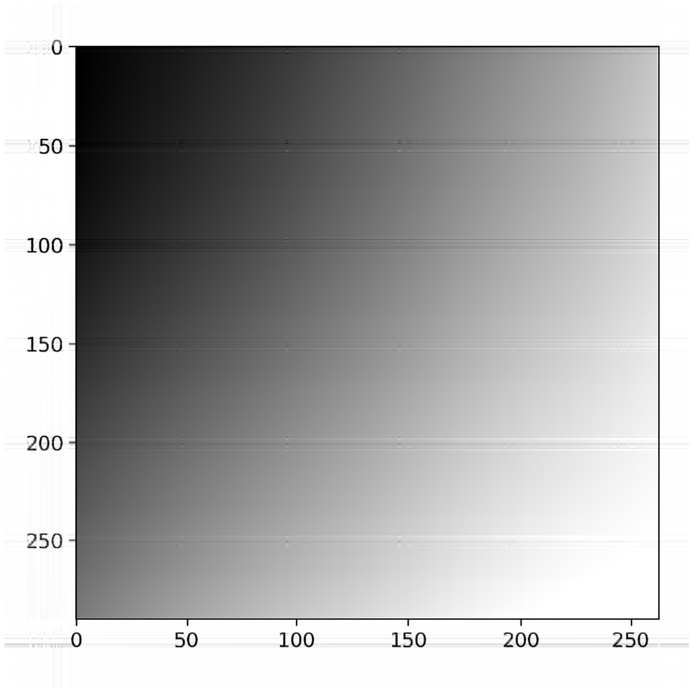

Figure 0: Image of globe at k= 500

Trade-off between k, image quality, and compression

- As the value of the k increases, the Frobenious norm decreases
- As the value of k increases, the quality and the sharpness of the image increases but the compression ratio decreases
- As the value of k tends to n, the image obtained is almost similar to that of the original image

why only Randomized SVD using power iterations.?

When compared to other methods like jacobi transformation, the major differences are :

- Methods like Jacobi Transformation Computes the full set of singular values and vectors where as in the above it Computes only a few (truncated) dominant singular values/vectors.
- Jacobi Transformation is often used for dense matrices and it is less efficent for large matrices but in the above case it is highly efficient for large and sparse matrices.
- cost of the above algorithm to reconstruct the matrix using top k singular values is $O(mnk)$ where as jacobi transformation costs ($O(mk^2+nk^2)$) which is much more than the above algorithm costs
- Taking the above into consideration, Randomized SVD using power iterations can be considered as the best process

What if A is complex.?

- The above algorithm and the computations done above is applicable only if A is a real matrix
- As in the above case the matrix A is intensity matrix there is no chance of A being Complex matrix but let′s check what to do if A is complex
- include<complex.h> in the code
- Replace all double with double complex in the above library
- Use conjugate transpose instead of simple transpose

- use conj() instead of dot product in the Gram_schmidt function
- Now there will be 2 parts in the matrix one is matrix of real component and other is a matrix of imaginary component

References

- MIT OpenCourseWare – Gilbert Strang, SVD
- Low-rank approximation,wikipedia
- Randomized SVD using power iterations