Course Project (Software) : Image Compression Using Truncated SVD

AI25BTECH11001 - ABHISEK MOHAPATRA

CONTENTS

Summary of Gibert Strang's video

Each matrix $\mathbf{A}$ of order $m \times n$ can expressed in the form

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

where, $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrix of order $m \times m$ and $n \times n$, and

$$\mathbf{\Sigma} = diag\begin{pmatrix}\sigma_0 & \sigma_1 & \sigma_2 & ...\end{pmatrix}_{m \times n} \tag{2}$$

This can understand as basis vector of row space converted in column space

$$\mathbf{A}\mathbf{V} = \mathbf{U}\mathbf{\Sigma} \tag{3}$$

or,

$$\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i \tag{4}$$

And,$\mathbf{U}$ and $\mathbf{V}$ can be found by

$$\mathbf{A}\mathbf{A}^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^\top\mathbf{U}^\top \tag{5}$$

$$\mathbf{A}^\top\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^\top\mathbf{U}^\top \tag{6}$$

where σ_i^2 are the eigen values of $\mathbf{A}\mathbf{A}^\top$ and $\mathbf{A}^\top\mathbf{A}$

Explanation of the Implemented Algorithm

Block Power Method

Block Power method replies on Power method for finding the eigenvector with the largest eigen vector.

Power Method

→ Each vector can be expressed in terms of as a linear combination of eigen vectors (as basis vector).

$$\mathbf{v} = \mu_0\mathbf{v}_0 + \mu_1\mathbf{v}_1 + \mu_2\mathbf{v}_2 + \mu_3\mathbf{v}_3 + ... + \mu_{n-1}\mathbf{v}_{n-1} \tag{7}$$

where $\mathbf{v}_i$ are eigen vector to matrix.

→ So, if we multiply the matrix with the vector the eigen values will be multiplied with each corresponding iron vectors and if we keep on multiplying this the eigen value of the eigen vectors with the largest eigen values will dominant over other vectors and the sequence will converge to a vector parallel to the eigen vector with the largest eigen value.

$$\mathbf{A}^m\mathbf{v} = \mu_0\lambda_0^m\mathbf{v}_0 + \mu_1\lambda_1^m\mathbf{v}_1 + \mu_2\lambda_2^m\mathbf{v}_2 + \mu_3\lambda_3^m\mathbf{v}_3 + ... + \mu_{n-1}\lambda_{n-1}^m\mathbf{v}_{n-1} \tag{8}$$

$$\mathbf{A}^m\mathbf{v} = \mu_0\lambda_0^m\left(\mathbf{v}_0 + \frac{\mu_1\lambda_1^m}{\mu_0\lambda_0^m}\mathbf{v}_1 + \frac{\mu_2\lambda_2^m}{\mu_0\lambda_0^m}\mathbf{v}_2 + \frac{\mu_3\lambda_3^m}{\mu_0\lambda_0^m}\mathbf{v}_3 + ... + \frac{\mu_{n-1}\lambda_{n-1}^m}{\mu_0\lambda_0^m}\mathbf{v}_{n-1}\right) \tag{9}$$

$$\lim_{m\to\infty}\mathbf{A}^m\mathbf{v} = \mu_0\lambda_0^m\mathbf{v}_0 \tag{10}$$

→ First we will initialise a random matrix $\mathbf{V}$ of n vectors then we will multiply $\mathbf{A}^\top\mathbf{A}$ ($\mathbf{A}$ = given matrix) to each of this vector and then Orthonormalized and continue this to a certain number of this step.

→ By applying this method, we will generate a Matrix with mutually perpendicular vectors so that these vectors are unique singular vectors and correspond to a unique singular value. This is our $\mathbf{V}$ matrix.

Initially:

$$\mathbf{V} := \begin{pmatrix}\mathbf{v_0} & \mathbf{v_1} & \mathbf{v_2} & ... & \mathbf{v_{n-1}}\end{pmatrix} \tag{11}$$

where $\mathbf{v}_i$ are randomly (distinct to each other) unit vectors.

Iteration upto m:

$$\mathbf{V} := \mathbf{A}^\top\mathbf{A}\mathbf{V} \tag{12}$$

$$\mathbf{V} := orthonorm(\mathbf{V}) \tag{13}$$

Finally:

$$let \quad \mathbf{B} = \mathbf{AV} \tag{14}$$

$$U := orthonorm(\mathbf{B}) \tag{15}$$

And, let **C** be the product of the elementary operation performed on **B** to find **U**.

$$\Sigma \approx C \quad as \quad m \to \infty \tag{16}$$

or, we take the diagonal elements of C as other vanishs as m increases.

Compare Different Algorithms and Explain Why Did You Choose the Particular Algorithm

In comparision to other algorithms, this algorithm is :

- very easy to understand
- easy to implement
- this algorithm has some base to basis of svd (obtaining svd from $\mathbf{A}^\top\mathbf{A}$ and $\mathbf{A}\mathbf{A}^\top$)

But there are some other trade of such that it is:

- very slow then others
- numerically instability
- requires higher value of K for better image

I used it as it will increase the understanding of matrices in general and easy to implement.

Reconstructed images for different k and Error analysis

Overview

This report demonstrates **SVD-based image compression** on six images using low-rank approximation $A_k = U_k \Sigma_k V_k^T$. Each image is shown **with its compression results** for varying rank k. Metrics include:

- **Frobenius Error**: $||A - A_k||_F$
- **Relative Error**: $\frac{||A-A_k||_F}{||A||_F}$
- **Compression Ratio**: $\frac{mn}{k(m+n+1)}$
- **Time**: Full SVD + reconstruction

Einstein Image Compression

TABLE I: SVD Compression: `einstein.jpg`

k	Frob. Err.	Rel. Err.	Comp. Ratio	Time (s)
		Channel 1 (Red)		
2	58.41	0.496	45.87	0.014
5	49.46	0.420	18.35	0.017
20	32.46	0.275	4.59	0.041
50	21.40	0.182	1.83	0.095
100	9.80	0.083	0.92	0.220

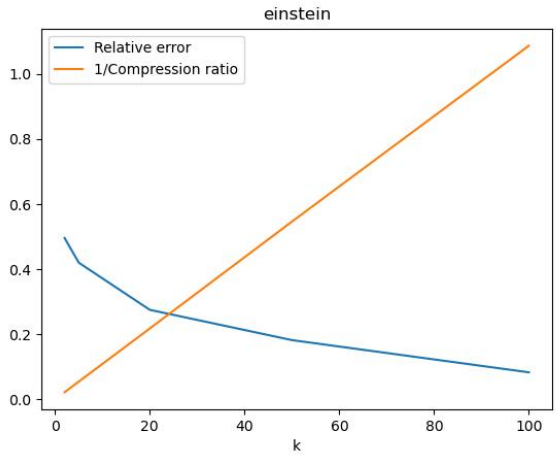
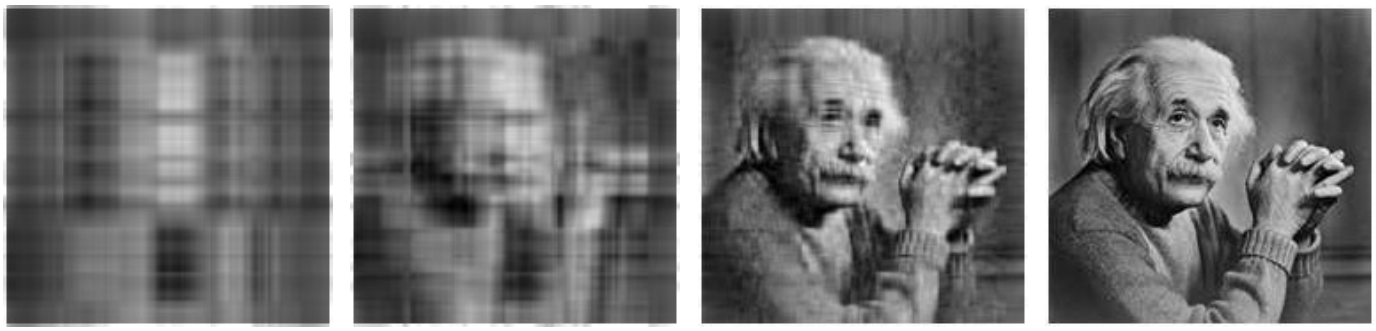


Fig. 1: Einstein compression results



(a) k=2 (b) k=5 (c) k=20 (d) k=100

Fig. 2: Einstein image at different compression levels (k = 2, 5, 20, 100)

Globe Image Compression

TABLE II: SVD Compression: `globe.jpg`

k	Frob. Err.	Rel. Err.	Comp. Ratio	Time (s)
		Channel 1 (Red)		
2	289.92	0.409	214.64	1.050
5	220.62	0.311	85.86	1.066
20	147.84	0.208	21.46	1.311
50	108.84	0.153	8.59	2.058
100	83.23	0.117	4.29	3.397
200	59.05	0.083	2.15	6.545
300	45.44	0.064	1.43	10.540
400	35.44	0.050	1.07	15.267
600	21.22	0.030	0.72	27.286

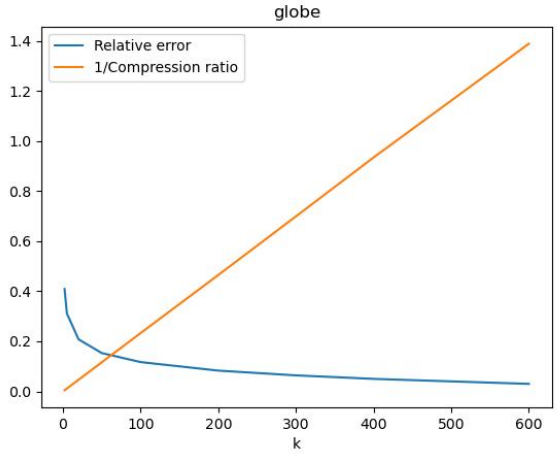


Fig. 3: Globe compression results

(a) k=1 (b) k=5 (c) k=10 (d) k=20 (e) k=50

Fig. 4: Globe image at different compression levels showing progressive quality improvement

Gauss Image Compression

TABLE III: SVD Compression: `gauss.jpg`

k	Frob. Err.	Rel. Err.	Comp. Ratio	Time (s)
		Channel 1 (Red)		
100	321.63	0.301	6.39	10.222
300	189.28	0.177	2.13	24.815
500	110.30	0.103	1.28	61.517

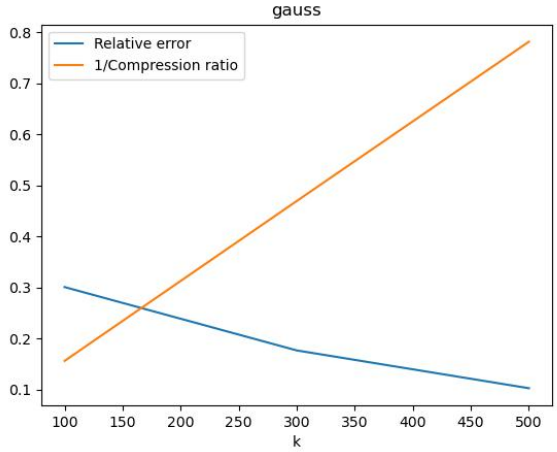


Fig. 5: Gauss compression results

(a) Gauss: k=100 (Low) (b) Gauss: k=300 (Medium) (c) Gauss: k=500 (High)

Fig. 6: Gauss portrait at different compression levels

Greyscale Image Compression

TABLE IV: SVD Compression: `greyscale.jpg`

k	Frob. Err.	Rel. Err.	Comp. Ratio	Time (s)
		Channel 1 (Grayscale)		
2	174.32	0.207	255.88	7.403
5	104.49	0.124	102.35	7.252
20	60.53	0.072	25.59	7.971
50	48.61	0.058	10.24	8.987
100	41.94	0.050	5.12	10.961
200	37.64	0.045	2.56	16.259
300	33.67	0.040	1.71	23.153
400	29.73	0.035	1.28	30.807
600	22.66	0.027	0.85	60.324

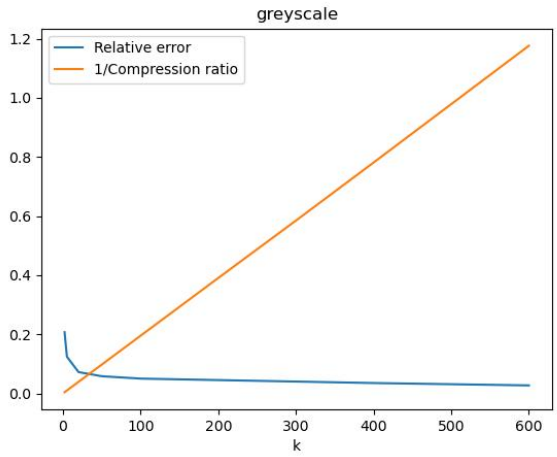
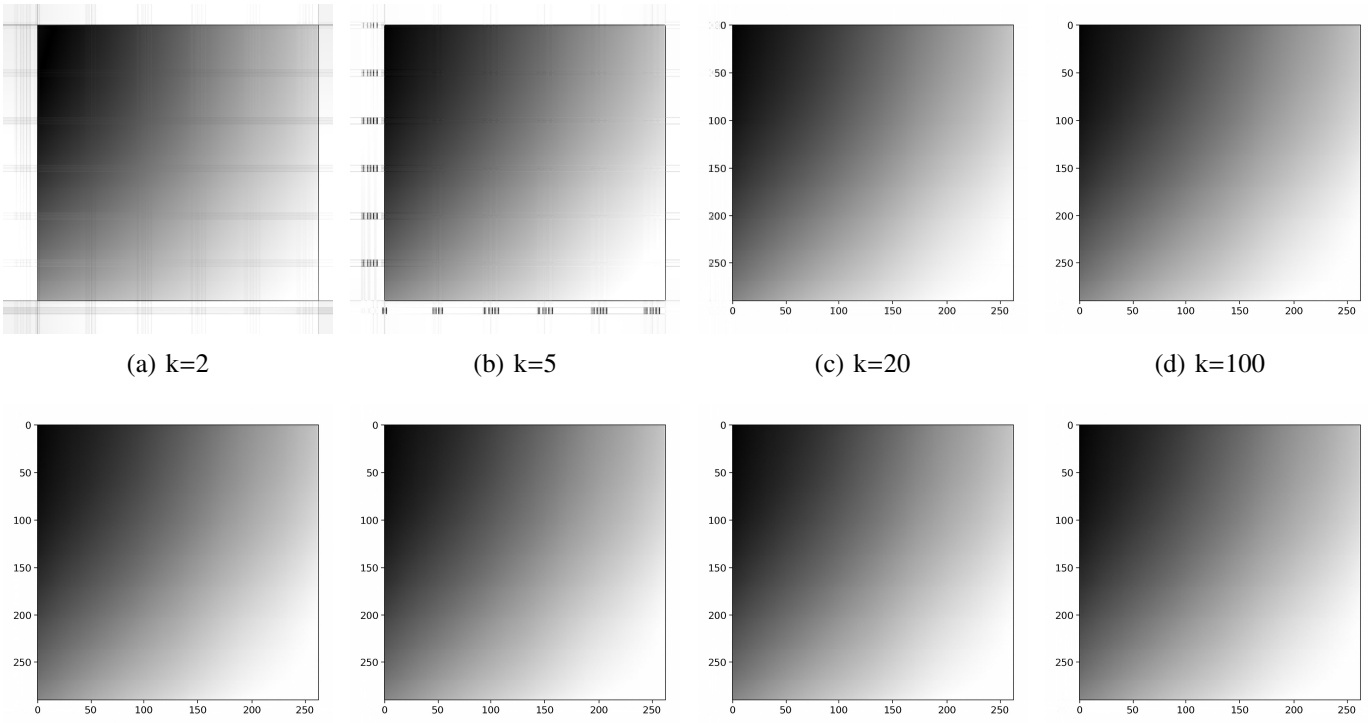


Fig. 7: Greyscale compression results



(a) k=2 (b) k=5 (c) k=20 (d) k=100

(e) k=200 (f) k=300 (g) k=400 (h) k=600

Fig. 8: Greyscale image at different compression levels (k = 2, 5, 20, 100, 200, 300, 400, 600)

Newton Image Compression (Color Channels)

TABLE V: SVD Compression: `newton.jpg` (All Channels)

k	Frob. Err.	Rel. Err.	Comp. Ratio	Time (s)
		Channel 1 (Red)		
10	154.77	0.359	41.78	2.847
50	125.52	0.291	8.36	5.498
100	115.51	0.268	4.18	9.265
300	84.76	0.196	1.39	30.138
500	56.35	0.131	0.84	61.889
		Channel 2 (Green)		
10	153.71	0.377	41.78	
50	125.18	0.307	8.36	
100	115.31	0.283	4.18	
300	84.79	0.208	1.39	
500	56.38	0.138	0.84	
		Channel 3 (Blue)		
10	145.58	0.428	41.78	
50	125.01	0.368	8.36	
100	115.33	0.339	4.18	
300	84.74	0.249	1.39	
500	56.37	0.166	0.84	

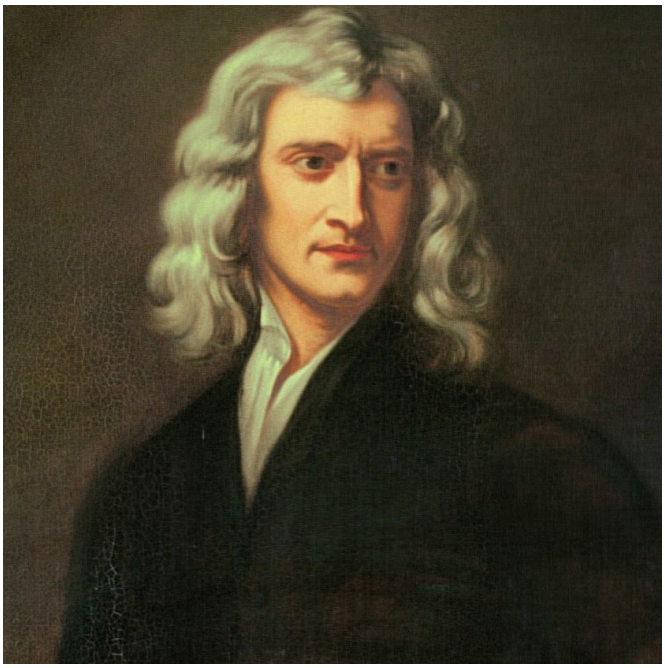


Fig. 9: Newton - k=100

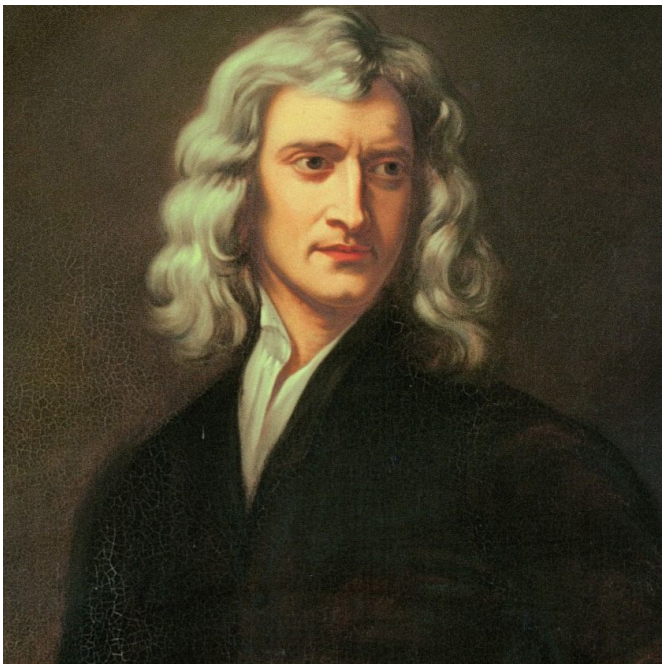


Fig. 10: Newton - k=300

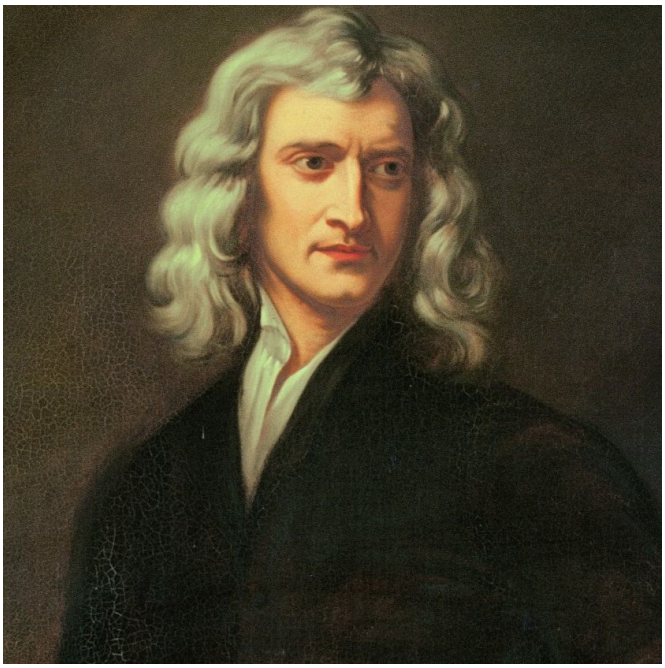


Fig. 11: Newton - k=500

Nebula Image Compression (Color Channels)

TABLE VI: SVD Compression: `nebula2.jpg` (All Channels)

k	Frob. Err.	Rel. Err.	Comp. Ratio	Time (s)
		Channel 1 (Red)		
100	221.05	0.340	6.40	37.377
300	150.64	0.232	2.13	83.269
500	104.24	0.160	1.28	189.603
		Channel 2 (Green)		
100	222.37	0.338	6.40	
300	151.50	0.231	2.13	
500	104.08	0.158	1.28	
		Channel 3 (Blue)		
100	215.96	0.356	6.40	
300	150.74	0.248	2.13	
500	104.02	0.171	1.28	

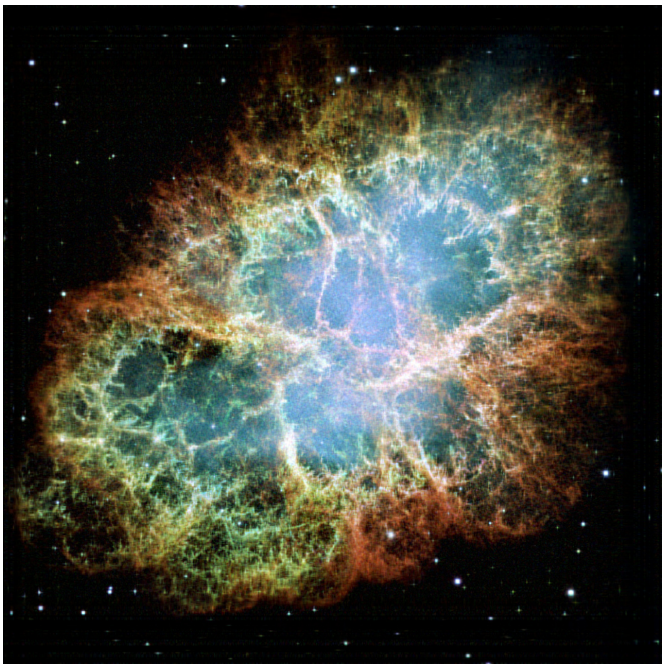


Fig. 12: Nebula2 - k=100

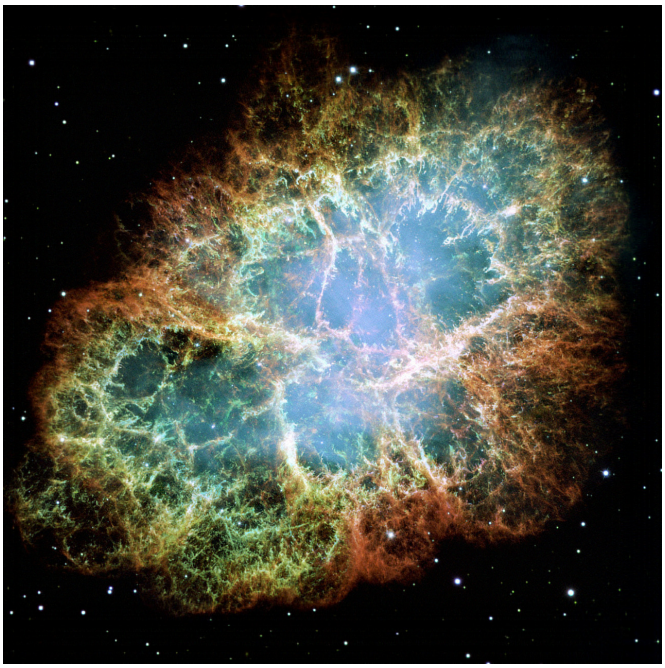


Fig. 13: Nebula2 - k=300

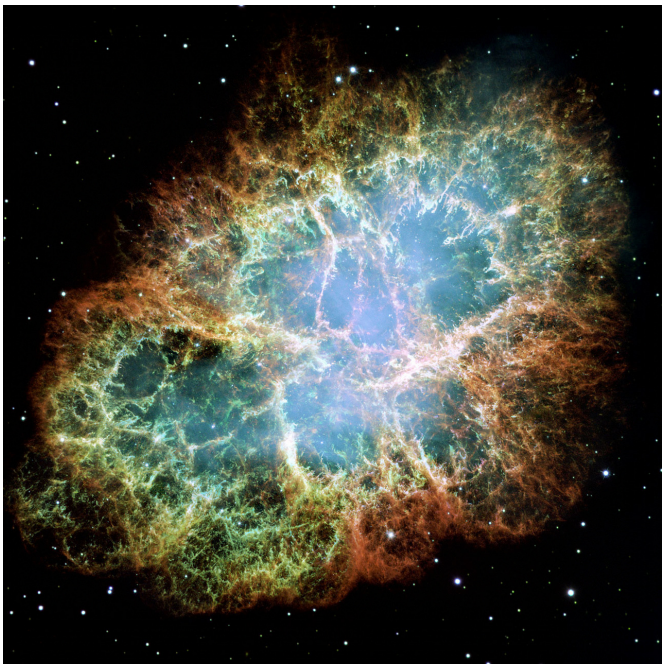


Fig. 14: Nebula2 - k=500

Discussion of Trade-offs and Reflections on Implementation Choice

Using the power block method we can get our images compressed in like 5 to 10 minutes but there are a few drawbacks to this method first it is very slow in comparison to other. Secondly it produce noisy images at low values of the k. Thirdly, this algorithm is also not perfect as it doesn't give the actual singular values correctly and even make mistakes in some of the lower values of k.

Methods like Golub Khan method which is very fast efficient and in the mean while is stable and does it produce very good images at lower value of k and is widely used.

Key Observations from Image Results

- **Einstein images**: Show clear progression from highly compressed (blocky at k=5) to recognizable features emerging at k=20, to good quality at k=50
- **Globe images**: Demonstrate the challenge of capturing fine details - even at k=50, some finer features remain blurred
- **Gauss portrait**: Similar to Einstein, shows good reconstruction quality starting around k=20 for facial features
- **Block Power Method limitations**: Visible as noise at lower k values, requiring higher k for acceptable quality compared to more other methods