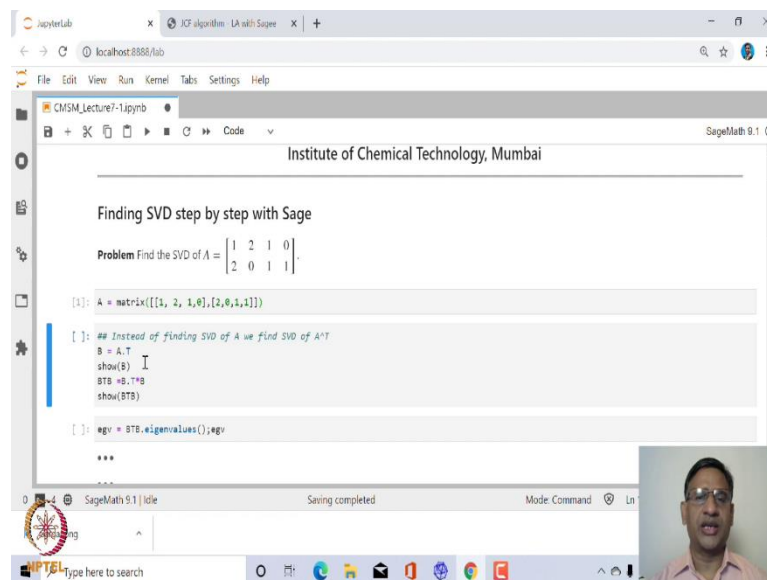


**Computational Mathematics with SageMath**  
**Prof. Ajit Kumar**  
**Department of Mathematics**  
**Institute of Chemical Technology, Mumbai**

**Lecture - 42**  
**Application of SVD to Image Processing**

Welcome to the 42nd lecture on Computational Mathematics with SageMath. In this lecture, we will look at an Application to Singular Value Decomposition to Image Processing. But, before that, let us look at how to find singular value decomposition step by step using SageMath.

(Refer Slide Time: 00:39)



```
CM/SM_Lecture7-1.ipynb
Institute of Chemical Technology, Mumbai

Finding SVD step by step with Sage

Problem Find the SVD of  $A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 2 & 0 & 1 & 1 \end{bmatrix}$ 

[1]: A = matrix([[1, 2, 1, 0], [2, 0, 1, 1]])

[2]: ## Instead of finding SVD of A we find SVD of A^T
B = A.T
show(B)
BTB = B.T*B
show(BTB)

[3]: egv = BTB.eigenvalues();egv

***
```

So, let us start with an example. So, suppose we have a matrix A which is 2 cross 4 matrix, we want to find its singular value decomposition, ok? So, first, let us define this matrix A. Now, instead of finding singular value decomposition of A, we will find singular value decomposition of A transpose, as A transpose is going to be 4 cross 2 matrix, and in that case, the matrix V in the singular value decomposition will be 2 cross 2. So, it will be easier for us to find eigenvalues and eigenvectors of 2 cross 2 matrix, ok?

(Refer Slide Time: 01:22)

```

[1]: A = matrix([[1, 2, 1, 1], [2, 0, 1, 1]])

[2]: # Instead of finding SVD of A we find SVD of A^T
    B = A.T
    show(B)
    BTB = B.T*B
    show(BTB)

[3]: egv = BTB.eigenvalues(),egv
[3]: [9, 3]
    ...
    ...

```

The output for `show(B)` is the matrix  $\begin{pmatrix} 1 & 2 \\ 2 & 0 \\ 1 & 1 \\ 0 & 1 \\ 6 & 3 \\ 3 & 6 \end{pmatrix}$ .

So, let us define B to be A transpose, and then define B transpose B as BTB. So, we need to find eigenvalue and eigenvectors of B transpose B. So, B transpose B is 6, 3, 3, 6 matrix. So, let us find eigenvalues of B transpose B, and the eigenvalues are 9 and 3. So, sigma 1 will be square root of 9, sigma 2 will be square root of 3.

(Refer Slide Time: 01:52)

```

[3]: [9, 3]

[4]: s1,s2 = sqrt(egv[0]),sqrt(egv[1])
    s1,s2

[4]: (3, sqrt(3))

[5]: S = matrix([[s1, 0, 0, 0], [0, s2, 0, 0]]).T
    show(S)

[5]: 
    ...
    ...

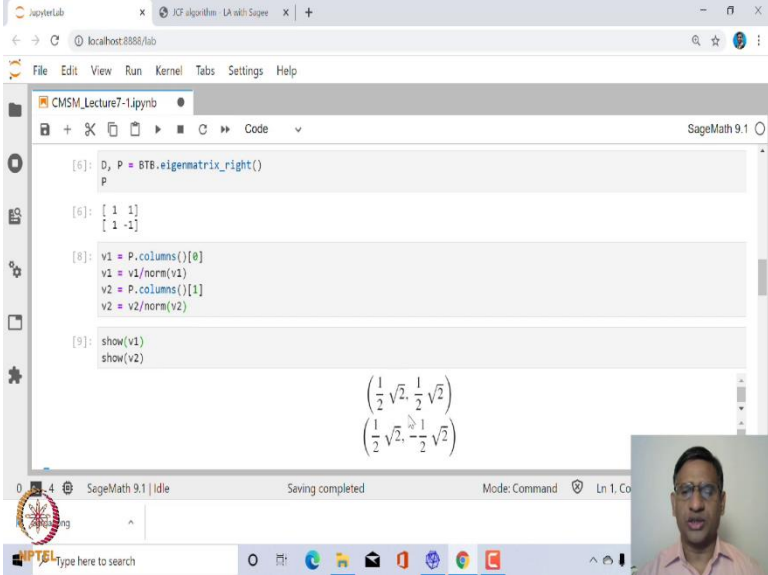
```

The output for `show(S)` is the matrix  $\begin{pmatrix} 3 & 0 \\ 0 & \sqrt{3} \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$ .

So, let us store this sigma 1 and sigma 2 in s1 and s2. So, s1 is 3, s2 is square root 3. Next, let us define this matrix S, or the matrix sigma. So, this sigma matrix is going to be, the first row will be s1, 0, 0, 0 and second row will be 0, s2, 0, 0 and we need to take transpose

of this, because this matrix B is 4 cross 2, therefore, the S in SVD will be of order 4 cross 2. So, this is the matrix sigma, ok?

(Refer Slide Time: 02:39)



```

[6]: D, P = BTB.eigenmatrix_right()
P
[6]: [ 1  1]
     [ 1 -1]

[8]: v1 = P.columns()[0]
v1 = v1/norm(v1)
v2 = P.columns()[1]
v2 = v2/norm(v2)

[9]: show(v1)
show(v2)

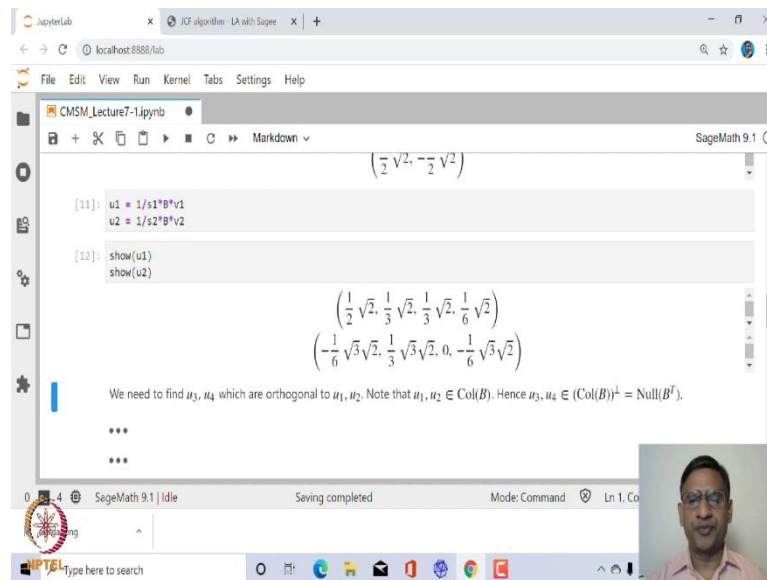
```

$$\begin{pmatrix} \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \\ \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} \end{pmatrix}$$

Next, we need to find eigenvalues, and eigen, eigenvalues we have already found of B transpose B. Next, we need to find eigenvectors. So, let us find eigen matrix under, underscore right, of B transpose B, and it gives you the, the diagonal matrix and the diagonalizing matrix P. So, here P is 1, 1, and 1 minus 1; that means, 1, 1, is eigenvector corresponding to eigenvalue 9, and 1, minus 1 is an eigenvector corresponding to eigenvalue 3, ok? So, let us store this in v1 and v2. So, first column of P, we will, we will store in v1. And second column of P, we will store in v2, and since we know we need v to be orthogonal matrix, we will divide v by its norm, and v1 by its norm, and v2 by its norm. So, this is what we get, and let us ask it to show what are v1 and v2.

So, v1 is square root 1 by 2 into square root 2, that is 1 by square root 2, 1 by square root 2, and v 2 is 1 by square root 2 minus 1 by square root 2. So, we have got this matrix V. V matrix, the first column will be square root 1 by square root 2, 1 by square root 2, second column will be 1 by square root 2, minus 1 by square root 2, right?

(Refer Slide Time: 04:10)

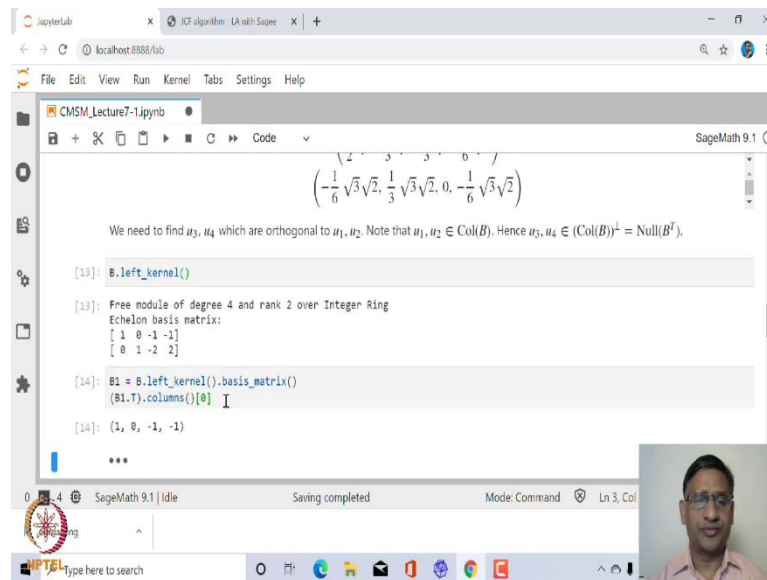


Now, we can define, once we have got  $v_1$  and  $v_2$ , we can define  $u_1, u_2$  of the matrix  $U$ , right?  $U$  here is going to be 4 cross 4 matrix. So,  $u_1$  will be 1 upon sigma 1, which is  $s_1$  here. So, 1 upon  $s_1$  times  $B$  times  $v_1$ ; similarly,  $u_2$  will be 1 upon  $s_2$  into  $B$  times  $v_2$ . It is easy to check why we are defining  $u_1$  and  $u_2$  like this, it is not difficult. So, if you just look at the way it is decomposed, this is how you will get, ok?

So, let us define  $u_1$  and  $u_2$ . So, we get  $u_1$  and  $u_2$  using, using  $v_1$  and  $v_2$ . So, we can find  $U$ , the first  $r$  columns of  $U$  by this method, because we cannot take reciprocal of 0. So, here, first  $r$  columns of  $U$ ,  $U$ , will obtained using this, where  $r$  is the rank of this matrix. So, these are  $u_1$  and  $u_2$ . So, let us ask it to show what are  $u_1$  and  $u_2$ . This is  $u_1$  and  $u_2$ , and you can.

So, next, we need to find what should be the other two columns of  $u$  namely,  $u_3$  and  $u_4$ . So, remember  $u_3$  and  $u_4$  has to be perpendicular to  $u_1$  and  $u_2$ , right? So, in particular,  $u_3$  and  $u_4$  has to lie in the, in the, in the orthogonal complement of this subspace, spanned by  $u_1$  and  $u_2$ , which is nothing but column space of  $B$ . Therefore,  $u_3$  and  $u_4$  will be in the orthogonal complement of the column space of  $B$ , but that is nothing but null, null space of  $B$  transpose, which is nothing but left kernel of  $B$ .

(Refer Slide Time: 06:17)



$$\left( -\frac{1}{6}\sqrt{3}\sqrt{2}, \frac{1}{3}\sqrt{3}\sqrt{2}, 0, -\frac{1}{6}\sqrt{3}\sqrt{2} \right)$$

We need to find  $u_3, u_4$  which are orthogonal to  $u_1, u_2$ . Note that  $u_1, u_2 \in \text{Col}(B)$ . Hence  $u_3, u_4 \in (\text{Col}(B))^\perp = \text{Null}(B^T)$ .

```

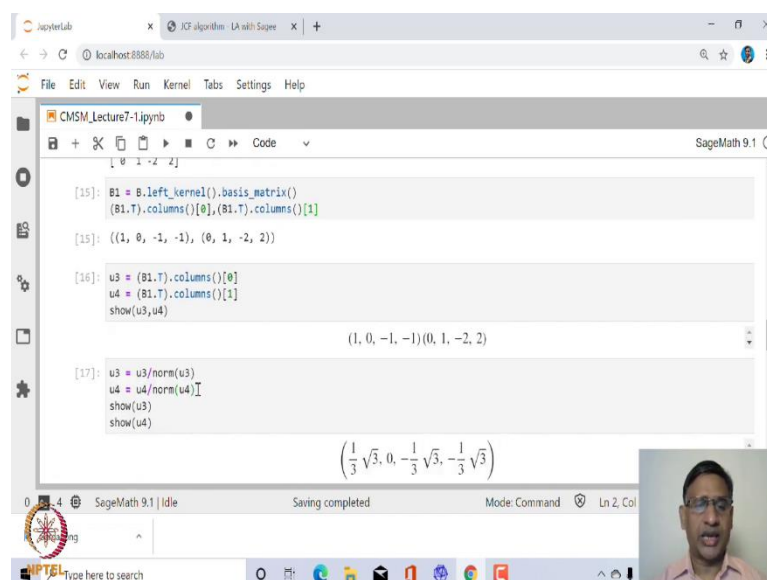
[13]: B.left_kernel()
[13]: Free module of degree 4 and rank 2 over Integer Ring
Echelon basis matrix:
[ 1  0 -1 -1]
[ 0  1 -2  2]

[14]: B1 = B.left_kernel().basis_matrix()
(B1.T).columns()[0]
[14]: (1, 0, -1, -1)

```

So, let us find this left kernel of the matrix B, we know that this is the left kernel, right? Next, let us store this vector, first vector in  $u_3$ , and the second vector, that is, second row of this, the basis matrix in  $u_4$ . So, let us, how do we do that? First let us find the basis matrix of the subspace of the left kernel, take the columns, take the transpose of this, take the first column, that will be, give you the first row.

(Refer Slide Time: 06:58)



```

[15]: B1 = B.left_kernel().basis_matrix()
(B1.T).columns()[0], (B1.T).columns()[1]
[15]: ((1, 0, -1, -1), (0, 1, -2, 2))

[16]: u3 = (B1.T).columns()[0]
u4 = (B1.T).columns()[1]
show(u3, u4)
[16]: (1, 0, -1, -1) (0, 1, -2, 2)

[17]: u3 = u3/norm(u3)
u4 = u4/norm(u4)
show(u3)
show(u4)
[17]: (1/3*sqrt(3), 0, -1/3*sqrt(3), -1/3*sqrt(3))

```

Similarly, if I take the second column, that will give you the second row of this, right? So, let us store these two columns in  $u_3$  and  $u_4$ . So, that is what we have done, and once you

have obtained  $u_3$  and  $u_4$ , this may not be orthogonal, orthonormal vectors. So, you need to divide by its norm.

(Refer Slide Time: 07:24)

```

[16]: u3 = (B1.T).columns()[0]
      u4 = (B1.T).columns()[1]
      show(u3,u4)

(1, 0, -1)(0, 1, -2)

[17]: u3 = u3/norm(u3)
      u4 = u4/norm(u4)
      show(u3)
      show(u4)

(1/3*sqrt(3), 0, -1/3*sqrt(3))
(0, 1/3, -2/3*sqrt(3))

```

So, let us divide by its norm and this is what you get. So, these are your  $u_3$  and  $u_4$ . You can check that this  $u_3$  and  $u_4$  are also orthogonal to each other, and not only that, this is perpendicular to  $u_1$  and  $u_2$ . If you notice here the way we defined, the way we defined  $u_1$  and  $u_2$  by this formula,  $\frac{1}{\sqrt{1+B_{11}}} \sum_{j=1}^n B_{1j} v_j$ , and this  $\frac{1}{\sqrt{1+B_{22}}} \sum_{j=1}^n B_{2j} v_j$ , you can directly check that  $u_1$  and  $u_2$  are orthogonal. In fact, they are orthonormal vectors. So, that is easy exercise.

(Refer Slide Time: 07:56)

```

[18]: U = column_matrix([u1,u2,u3,u4])
      V = column_matrix([v1,v2])

[19]: show(U)
      show(S)
      show(V)

```

$$U = \begin{pmatrix} \frac{1}{3}\sqrt{2} & -\frac{1}{6}\sqrt{3}\sqrt{2} & \frac{1}{3}\sqrt{3} & 0 \\ \frac{1}{3}\sqrt{2} & \frac{1}{3}\sqrt{3}\sqrt{2} & 0 & \frac{1}{3} \\ \frac{1}{3}\sqrt{2} & 0 & -\frac{1}{3}\sqrt{3} & -\frac{2}{3} \\ \frac{1}{6}\sqrt{2} & -\frac{1}{6}\sqrt{3}\sqrt{2} & -\frac{1}{3}\sqrt{3} & \frac{2}{3} \end{pmatrix}$$

$$S = \begin{pmatrix} 3 & 0 \\ 0 & \sqrt{3} \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$V = \begin{pmatrix} \frac{1}{3}\sqrt{2} \\ \frac{1}{3}\sqrt{2} \end{pmatrix}$$

Now, let us define the matrix U, which is column matrix of u1, u2, u3, u4, and matrix V as column matrix of v1, v2, v3, sorry, v1 and v2. There are two vectors only, right? Now, you can ask it to show what are U? So, this is U.

(Refer Slide Time: 08:16)

```

show(V)

```

$$U = \begin{pmatrix} \frac{1}{3}\sqrt{2} & -\frac{1}{6}\sqrt{3}\sqrt{2} & \frac{1}{3}\sqrt{3} & 0 \\ \frac{1}{3}\sqrt{2} & \frac{1}{3}\sqrt{3}\sqrt{2} & 0 & \frac{1}{3} \\ \frac{1}{3}\sqrt{2} & 0 & -\frac{1}{3}\sqrt{3} & -\frac{2}{3} \\ \frac{1}{6}\sqrt{2} & -\frac{1}{6}\sqrt{3}\sqrt{2} & -\frac{1}{3}\sqrt{3} & \frac{2}{3} \end{pmatrix}$$

$$S = \begin{pmatrix} 3 & 0 \\ 0 & \sqrt{3} \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$V = \begin{pmatrix} \frac{1}{3}\sqrt{2} \\ \frac{1}{3}\sqrt{2} \end{pmatrix}$$

And, this is the sigma, and this is the V.

(Refer Slide Time: 08:19)

$$B = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \\ \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} \end{pmatrix}$$

```

[20]: U*S*V==B
[20]: True

```

Therefore, we have obtained, we have obtained singular value decomposition of B, right? So, you can check that U into S into V should be equal to B.

(Refer Slide Time: 08:36)

```

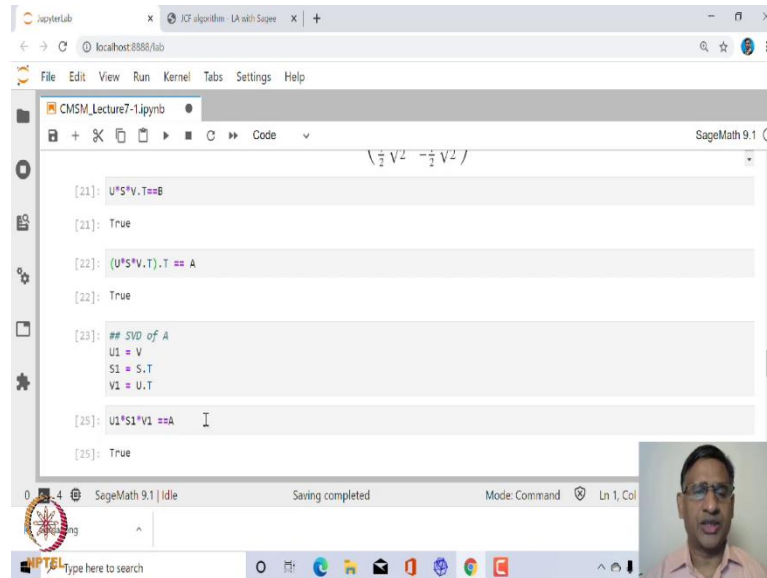
[21]: U*S*V.T==B
[21]: True
[22]: (U*S*V.T).T == A
[22]: True
[23]: ## SVD of A
      U1 = V
      S1 = S.T
      V1 = U.T
[24]: U1*S1*V1.T == A
[24]: False

```

Next, actually this all should have been V transpose, but V, in this case, is a, is symmetric. So, V transpose will be itself. Now, you can also check that if I take the transpose of this U into S into V transpose, that you should get as A. So, therefore, singular value, we have obtained singular value decomposition of A. So, this is going to be V into S transpose into U transpose. So, if you want, you can call this as U1, as U1 as V transpose U, U1 will be

V, in this case, and U. S 1 will be S transpose, and V1 will be U transpose, and then when you multiply U1 S1 V1, this will be transpose, you should get this as, I think it is to, B, right?

(Refer Slide Time: 09:31)



```

[21]: U*S*V.T==B
[21]: True

[22]: (U*S*V.T).T == A
[22]: True

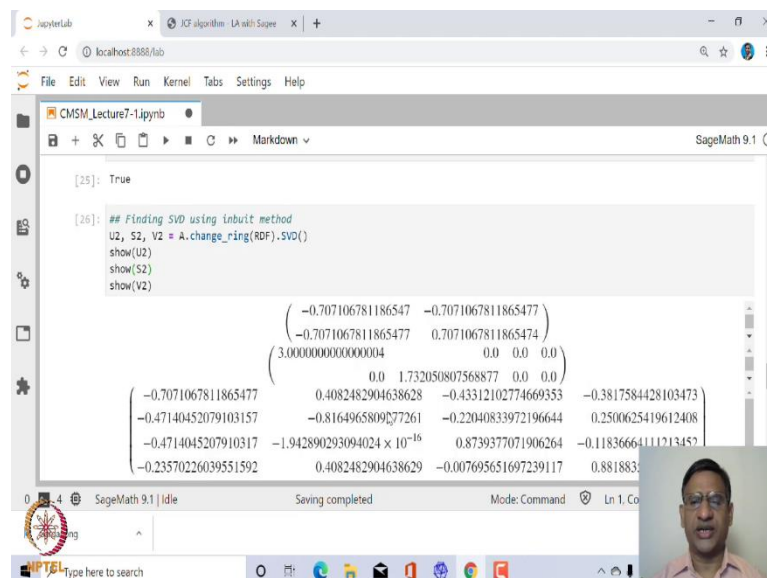
[23]: ## SVD of A
      U1 = V
      S1 = S.T
      V1 = U.T

[25]: U1*S1*V1 == A
[25]: True

```

So, this is what you, you get, right?

(Refer Slide Time: 09:39)



```

[26]: ## Finding SVD using inbuilt method
      U2, S2, V2 = A.change_ring(RDF).SVD()
      show(U2)
      show(S2)
      show(V2)

```

$$\begin{pmatrix} -0.707106781186547 & -0.707106781186547 \\ -0.707106781186547 & 0.707106781186547 \\ 3.00000000000000 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

$$\begin{pmatrix} -0.707106781186547 & 0.4082482904638628 & -0.43312102774669353 & -0.3817584428103473 \\ -0.47140452079103157 & -0.8164965809277261 & -0.22040833972196644 & 0.2500625419612408 \\ -0.4714045207910317 & -1.942890293094024 \times 10^{-16} & 0.8739377071906264 & -0.11836664111213452 \\ -0.23570226039551592 & 0.4082482904638629 & -0.007695651697239117 & 0.881883 \end{pmatrix}$$

So, this is how you can find step by step this SVD decomposition, but you can also verify this using inbuilt function. So, let us change the ring of A to RDF so that this

decomposition will work, and find the SVD decomposition and store this in U1, S1, V2, and then let us look at what are these matrix.

Of course, these, these will be in decimal numbers, what we have found, obtained in square root, but you can check that this is approximation of what we have got, right? So, this is how you can find a singular value decomposition of a matrix step by step, but of course, this, this you will be able to do mostly when you have a very small, small matrices. If it is a very big matrix, during this computation will be quite challenging manually, right? So, of course, using Sage you can always execute this, right?

(Refer Slide Time: 10:35)

Application of SVD to Image Processing

Let

$$A_0 = U \sum V^T$$

where Let  $U = [u_1 \ u_2 \ \dots \ u_m]$ ,  $V = [v_1 \ v_2 \ \dots \ v_n]$  and

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_r & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix}$$

Now, let us look at an application to singular value decomposition to image processing, basically image compression, right? So, let us recall that if A has singular value decomposition as U into sigma into V transpose, where U we will write as u 1, u 2,..., u m and V we will write as column vectors v 1, v 2,..., v n.

And, here, sigma is this matrix, which is, suppose A is m by n matrix, this, this sigma will be m cross n matrix of this form, where sigma 1, sigma 2,..., sigma r are singular values of A, and with the condition that sigma 1 is bigger than equal to sigma 2, sigma 2 is bigger than equal to sigma 3, and so on, right? So, we are assuming rank of A to be r.

(Refer Slide Time: 11:23)

The screenshot shows a JupyterLab window with a SageMath 9.1 notebook titled 'CMSM\_Lecture7-1.ipynb'. The notebook content is as follows:

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_r & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix}$$

$$A = U \Sigma V^T = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \dots + \sigma_r u_r v_r^T$$

with  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r$ . This is known as rank one decomposition of the matrix.

$$\sum_{i=1}^k \sigma_i u_i v_i^T \approx A$$

as  $k \rightarrow r$ .

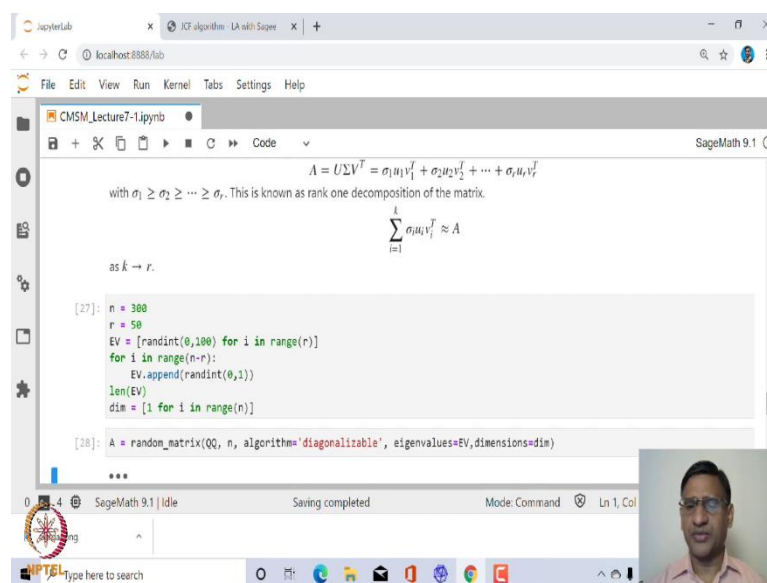
\*\*\*

The interface also shows a status bar at the bottom with 'SageMath 9.1 | Idle', 'Saving completed', 'Mode: Command', and 'Ln 2, Col 1'. A small video inset of a man is visible in the bottom right corner.

Now, if you, if you just multiply this, U, S into U sigma into V, this is how you, you will be able to get this. So, A is equal to sigma 1 times u 1 into v 1 transpose, sigma 2 times u 2 into v 2 transpose, plus dot dot dot, sigma r into u r into v r transpose. If you notice, u 1 and u 2 both are column vectors, therefore, both will have, and they are orthogonal matrices, therefore, it cannot be non zero. Therefore, each u 1 and v 1 is, similarly, u i and v i will have rank 1, and therefore, this multiplication will also have rank 1. So, A has become sum of rank 1 matrices namely, r of them and that is why the rank of A is r, right?

Therefore, suppose we, we take only first k components of this sum on the right-hand side, this is sigma i u i into v i transpose, i going from 1 to k, and as k goes to r, this, this left-hand side will be, will be equal to A, right? So, this is. So, this we can think of as an approximation of this matrix A, right, as, as a rank 1, approximation sum of rank 1 matrices. Now, what happens is that, we know that sigma 1 is the largest, followed by sigma 2 and sigma 3. So, in case if we have very large matrix, it is possible that last few sigma i's will be very close to 0, and so, therefore, most of the information about A are contained inside first few elements of this sum, ok? So, that simply means that you can ignore last few components in this. So, first few components itself will be very good approximation of A. So, let, this is what is used in image compression. So, let us see how we can, we can use this in order to approximate, or compress an image.

(Refer Slide Time: 13:38)



The screenshot shows a JupyterLab window with a SageMath 9.1 notebook. The notebook has two cells. The first cell contains mathematical text and a formula:

$$A = U \Sigma V^T = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \dots + \sigma_r u_r v_r^T$$

with  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r$ . This is known as rank one decomposition of the matrix.

$$\sum_{i=1}^k \sigma_i u_i v_i^T \approx A$$

as  $k \rightarrow r$ .

The second cell contains the following code:

```
[27]: n = 300
      r = 50
      EV = [randint(0,100) for i in range(r)]
      for i in range(n-r):
          EV.append(randint(0,1))
      len(EV)
      dim = [1 for i in range(n)]

[28]: A = random_matrix(QQ, n, algorithm='diagonalizable', eigenvalues=EV, dimensions=dim)

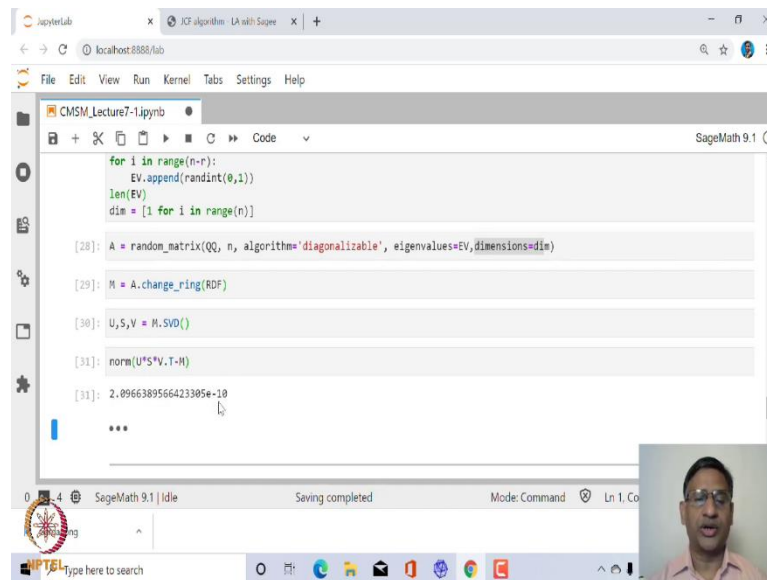
***
```

The bottom status bar shows 'SageMath 9.1 | Idle', 'Saving completed', 'Mode: Command', and 'Ln 1, Col 1'. A video feed of a person is visible in the bottom right corner.

So, before that, let us just take an example. We, we have generated a random matrix, which is 3, 300 by 300 and in, in such a way that first r eigen, in, in such a way that first r eigenvalues, values, will be random number between 0 and 100, and remaining r could be 0 and 1. The remaining n minus r could be 0 and 1, ok? So, there there is a inbuilt function called random underscore matrix, with which you can generate a random matrix over ZZ or QQ, and this could be diagonalizable matrix. So, here this is diagonalizable matrix, and whose eigenvalues are this some r sorry, n random numbers.

And, what we have done is, we have said that the, the last, let us say, first 50 are random, first 50 eigenvalues are some random numbers, random integers between 0 and 100, remaining r could be 0 or 1. So, lots of eigenvalues will be 0 in this case. So, let us, let us run this. So, this has an inbuilt method to generate a diagonalizable matrix with eigenvalues mentioned in that. Of course, you should, also need to mention what should be the, the multiplicity, algebraic multiplicity of each of this eigenvalue. So, this, this, in this case, it says that algebraic multiplicity of each of this eigenvalue is 1, right? So, therefore, it is diagonalizable, right?

(Refer Slide Time: 15:27)



The screenshot shows a JupyterLab window with a SageMath 9.1 kernel. The code in the cell is as follows:

```
for i in range(n-r):
    EV.append(randint(0,1))
len(EV)
dim = [1 for i in range(n)]

[28]: A = random_matrix(QQ, n, algorithm='diagonalizable', eigenvalues=EV, dimensions=dim)

[29]: M = A.change_ring(RDF)

[30]: U, S, V = M.SVD()

[31]: norm(U*S*V.T-M)

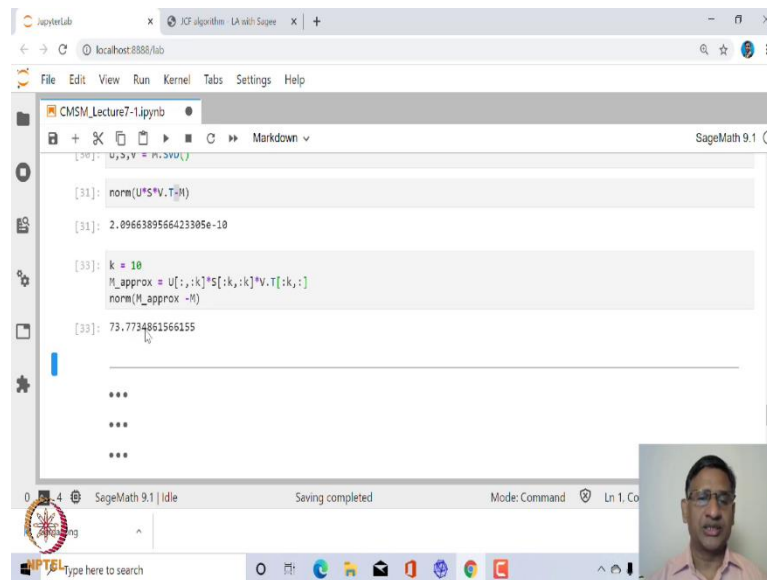
[31]: 2.0966389566423305e-10
***
```

The output shows the norm of the difference between the matrix  $U \cdot S \cdot V^T$  and the original matrix  $M$  is approximately  $2.0966389566423305 \times 10^{-10}$ . A small video inset of a person is visible in the bottom right corner of the JupyterLab window.

Now, let us change the ring of A to RDF, so that we can find its eigen, singular value decomposition. Now, let us find singular value decomposition of this M, which is the A changed over RDF. Next, let us find out what is the difference between the matrix A, and the matrix S into, U into S into V transpose, singular value decomposition of M.

So, the length of this should be very close to 0. Since it is a numerical approximation, numerical way of computing this singular value decomposition, you may not get this difference to be, norm of this difference to be 0 exactly, but it will be very close to 0. So, it says that it is of the order 10 to the power minus 10.

(Refer Slide Time: 16:24)



```
[10]: U, S, V = P1.SVD(U)
```

```
[31]: norm(U*S*V.T, H)
```

```
[31]: 2.0966389566423305e-10
```

```
[33]: k = 10
```

```
      M_approx = U[:, :k]*S[:, :k]*V.T[:, :k, :]
```

```
      norm(M_approx - M)
```

```
[33]: 73.7734861566155
```

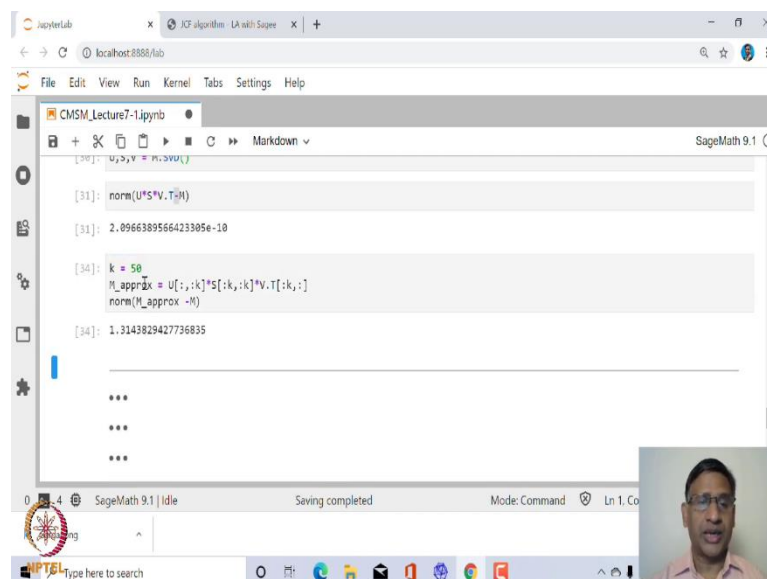
\*\*\*

\*\*\*

\*\*\*

Now, suppose instead of taking all the terms, all the  $r$  terms here, we take, for example, only first, let us say few terms. So, let us take only first 10 terms. So, first 10 terms, how will you obtain first 10 terms? You take first all the rows, and first  $k$  columns of  $U$ , then take first  $k$  rows and  $k$  columns of  $S$ , and then take only first  $k$  rows of  $V$  transpose, ok? So, that is, and here  $k$  is 10. So, we are taking first 10 elements in this sum, right? So, let us see in this case what is the, the difference between the norm of the original matrix  $M$  and this matrix. So, this, in this case it is 73.77 which is quite large as an error.

(Refer Slide Time: 17:20)



```
[10]: U, S, V = P1.SVD(U)
```

```
[31]: norm(U*S*V.T, H)
```

```
[31]: 2.0966389566423305e-10
```

```
[34]: k = 50
```

```
      M_approx = U[:, :k]*S[:, :k]*V.T[:, :k, :]
```

```
      norm(M_approx - M)
```

```
[34]: 1.3143829427736835
```

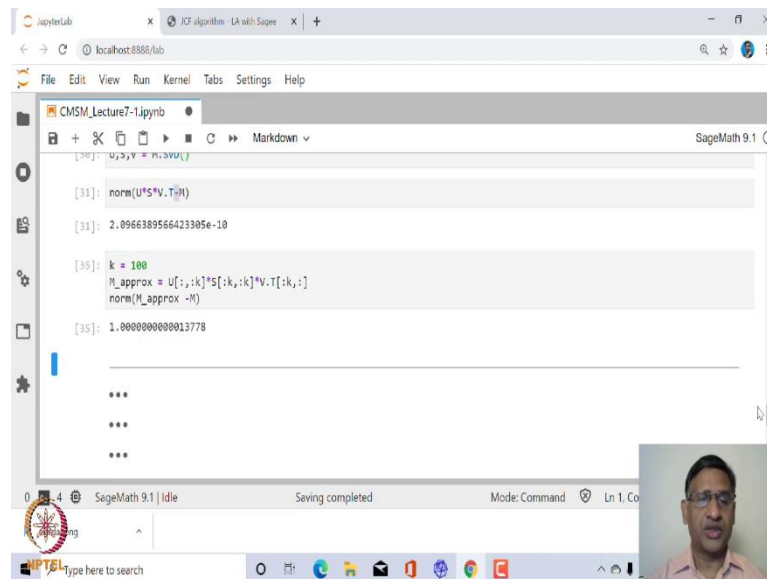
\*\*\*

\*\*\*

\*\*\*

But, instead of 10 if I make it 50, then it has reduced considerably, right?

(Refer Slide Time: 17:30)



```
[31]: norm(U*S*V.T*H)
[31]: 2.0966389566423305e-10

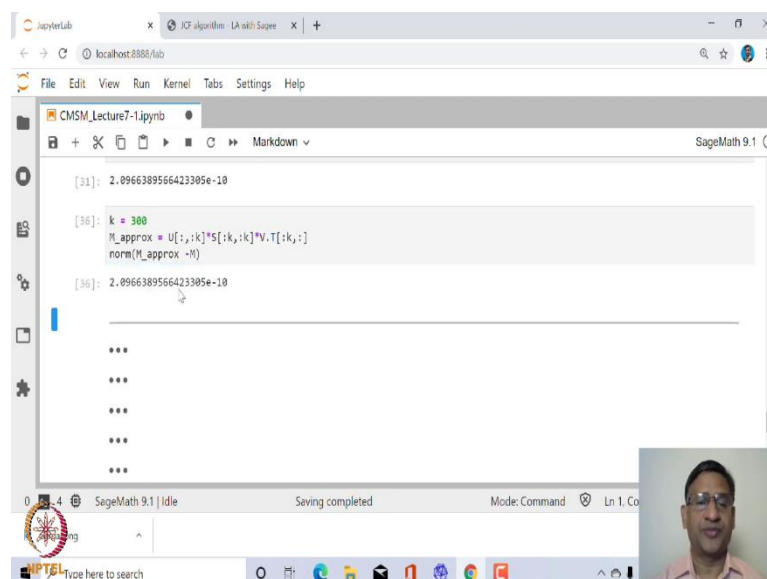
[35]: k = 100
M_approx = U[:, :k]*S[:, k]*V.T[:, k, :]
norm(M_approx - H)
[35]: 1.0000000000013778

...
...
...
```

If I take instead of 50, if I take 100, it will be again very close to 1, right? So, in this case, because lot of entries in this matrix is going to be almost close to 0, so, that is why you are able to. So, what it means is that first 100 terms in this matrix which, what is the order of this matrix? 5, 300 cross 300.

So, if you just take first 300 terms of this matrix you are getting very good approximation. If I have taken all of them, let us say 300, you will get again close to 0, right?

(Refer Slide Time: 18:04)

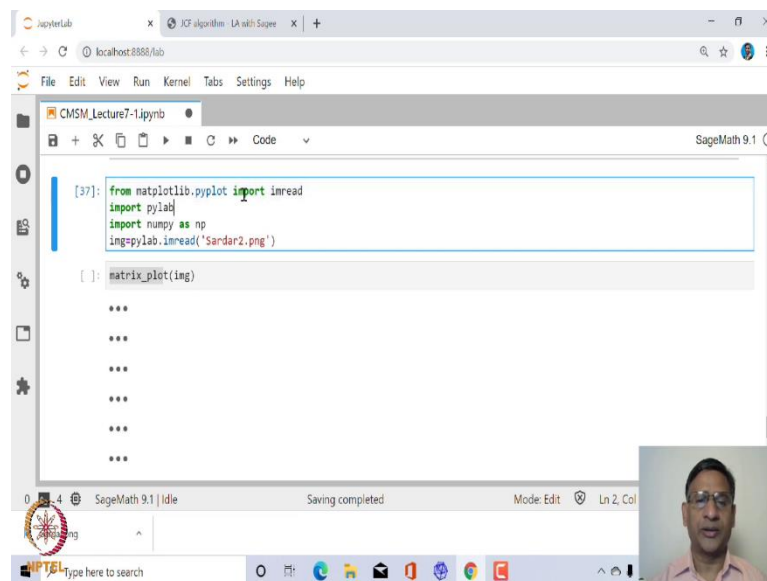


```
[31]: 2.0966389566423305e-10

[36]: k = 300
M_approx = U[:, :k]*S[:, k]*V.T[:, k, :]
norm(M_approx - H)
[36]: 2.0966389566423305e-10

...
...
...
...
...
```

So, this is what we will use in case of compressing an image. So, let us take an example.  
(Refer Slide Time: 18:16)



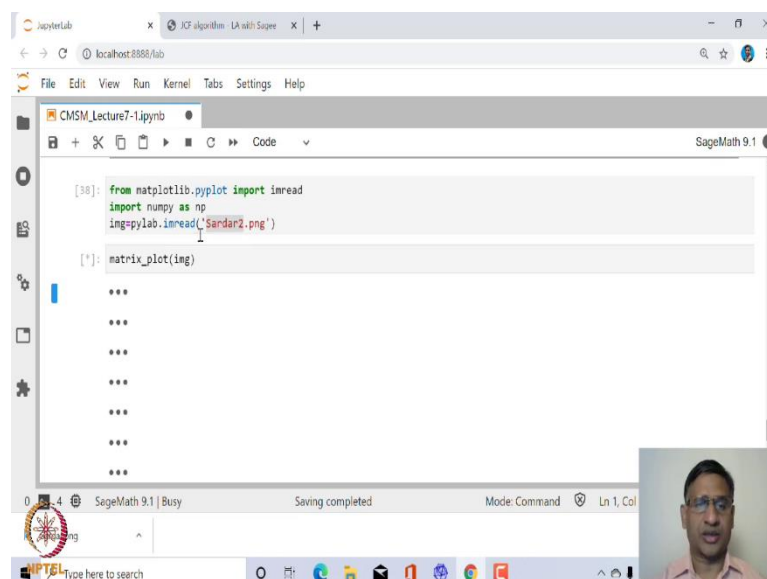
```
[37]: from matplotlib.pyplot import imread
import pylab
import numpy as np
img=pylab.imread('Sardar2.png')

matrix_plot(img)

***
***
***
***
***
***
```

So, let us import an image, let me plot. So, the SageMath has a function called matrix underscore plot. So, using which, you can plot any matrix. So, what we have done here is, actually this is just a python command. So, from matplotlib pyplot, we have imported a function called image read, and then import pylab, and then inside pylab there is a function called image read; actually this is not required. This is not required.

(Refer Slide Time: 18:51)



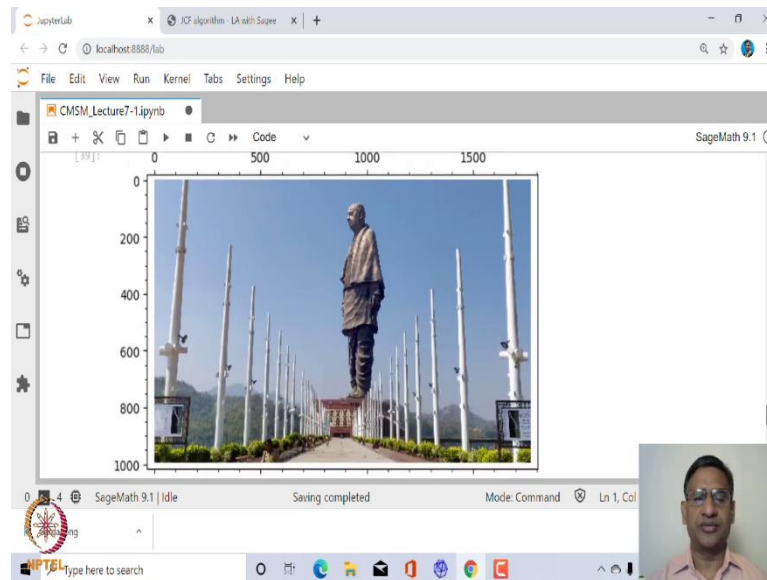
```
[38]: from matplotlib.pyplot import imread
import numpy as np
img=pylab.imread('Sardar2.png')

matrix_plot(img)

***
***
***
***
***
***
```

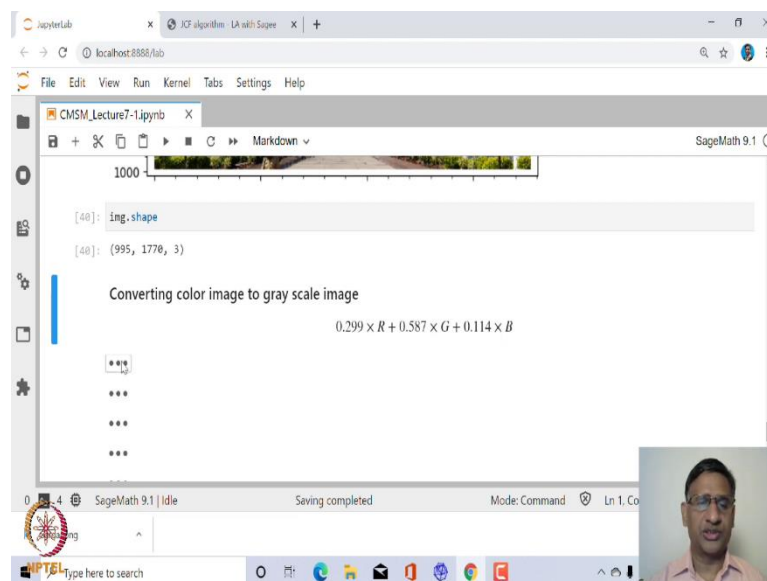
And, then we are reading an image, the name I have given is Sardar2 dot png. This is image of Statue of Unity, right? So, when we plot this, so, what it has done? It has converted this into, ok. So, this is the image we have, right?

(Refer Slide Time: 19:11)



Now, what we are going to do is, we will convert this image into a matrix. So, how do we do that?

(Refer Slide Time: 19:21)

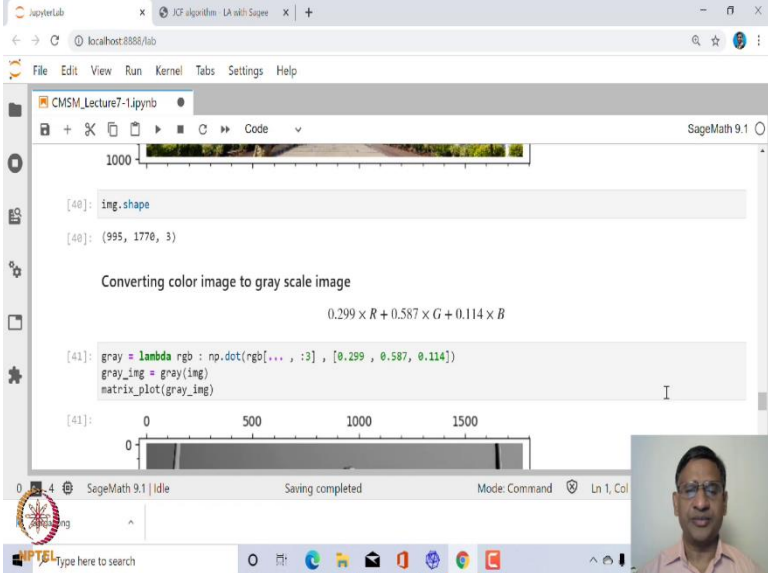


So, in order to, first, before that, let us look at what is the order of this matrix. The order of this matrix is 995 by 1770; that is quite large matrix, that is very big matrix, because

this, this image has very good resolution, and this, this, this third component, that gives you the color. So, this is a color image, but we have done singular value decomposition of the  $m$  by  $n$  matrix.

So, we, we need to convert this into gray image, so that this color component we do not have. So, now, what do we do? We convert, there are several ways in which one can convert this color image into gray image. There are several inbuilt methods, but this is the transformation generally one uses. So, red component, take 29 percent, 29 percent, 9 percent green component, take this, and this is the blue component.

(Refer Slide Time: 20:18)



```
[40]: img_shape
[40]: (995, 1778, 3)

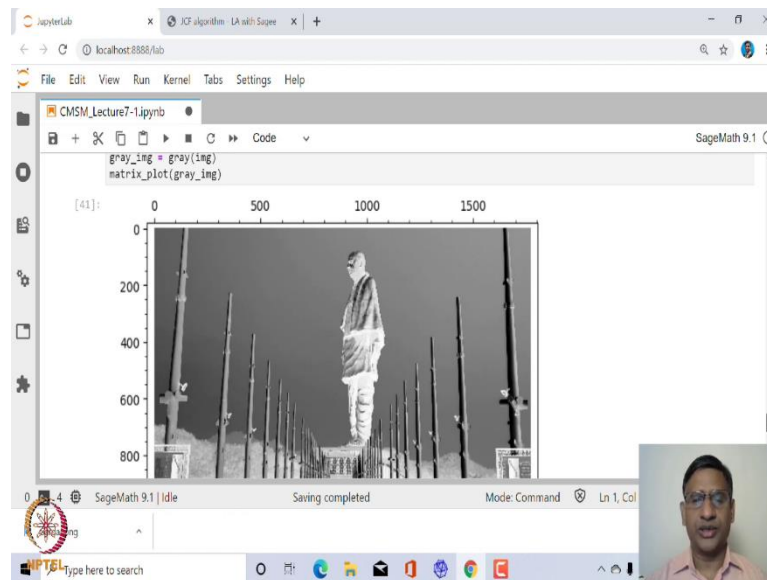
Converting color image to gray scale image


$$0.299 \times R + 0.587 \times G + 0.114 \times B$$


[41]: gray = lambda rgb : np.dot(rgb[... , :3] , [0.299 , 0.587 , 0.114])
      gray_img = gray(img)
      matrix_plot(gray_img)
[41]:
```

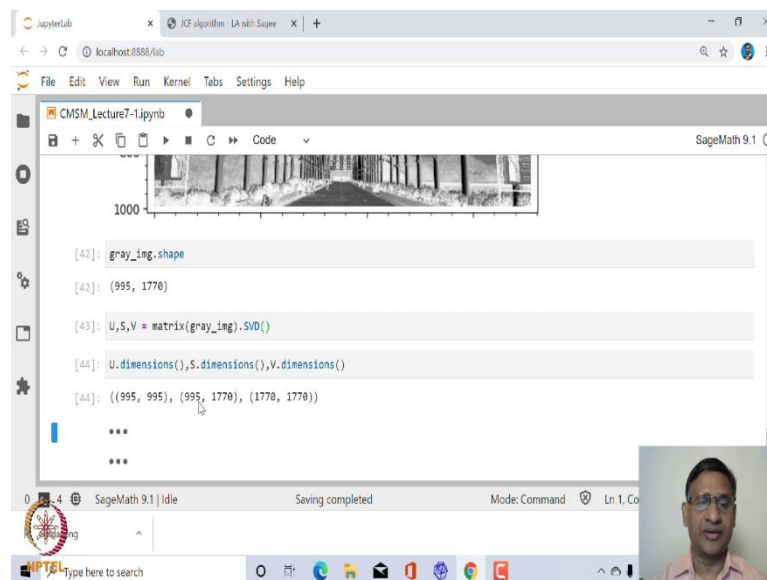
So, this is what we will use to convert this image into gray image, and this is what you have.

(Refer Slide Time: 20:24)



Now, this we, image is converted into gray image. It does not look all that grayed, but however, it will retain the dimension.

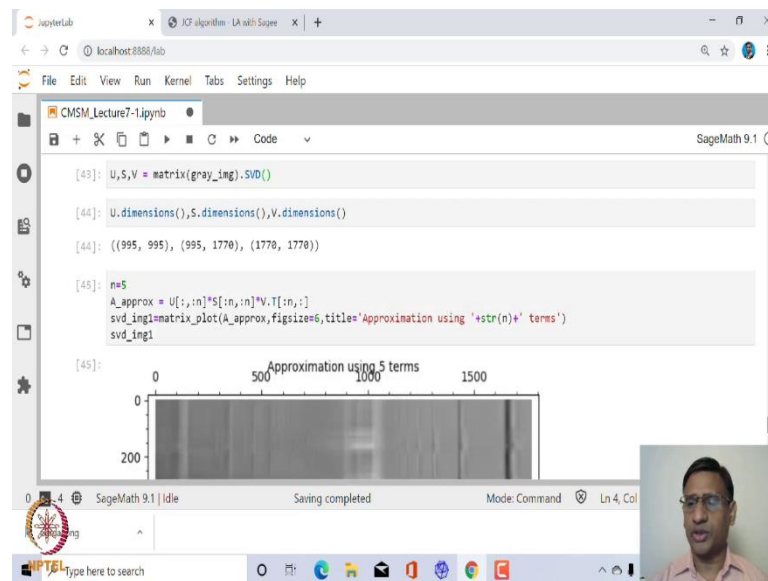
(Refer Slide Time: 20:35)



So, now, let us look at what is the, the shape of this image. So, it is 995 by 1770. Now, what we will do? We will find singular value decomposition of this matrix. This is very large matrix, and if you find singular value decomposition, it may take some, some time, may be few seconds, depending upon the speed of your computer, right?

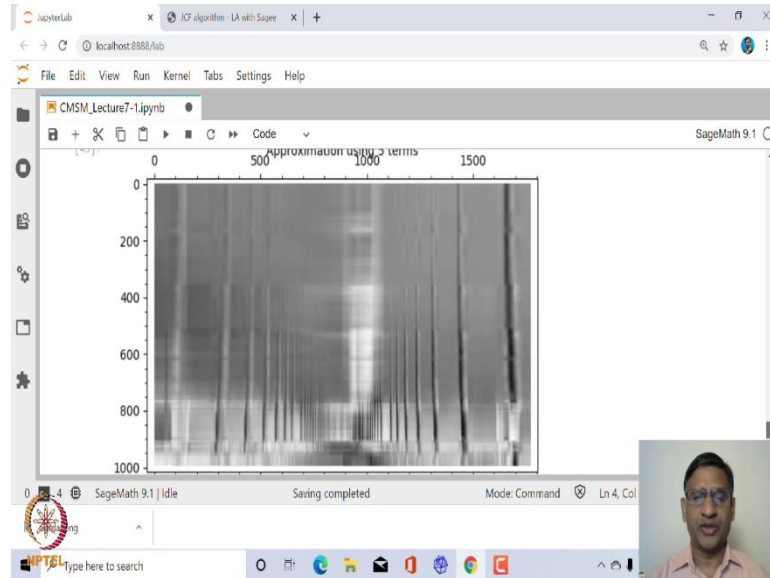
So, it has done. It did not take too much of time. So, it has found out singular value decomposition of this matrix, which is the, the matrix associated with this image. Image consists of pixels, each pixels can be thought of as a matrix entry, right? So, now you can find out what should be the dimension of each of this U, S and V. U has 90, 999, 995 cross 995, S is 995 cross 1770 and V is 1770 cross 1770, right?

(Refer Slide Time: 21:30)



Now, let us approximate, let us take only first 5 terms of this, sum of this, this sum, only first 5 terms of this. So, that will be an approximation of A using first 5 terms, and then let us try to plot graph of that approximation, ok? So, let me get rid of this. So, if I take only first 5 terms of this approximation. So, you are taking first 5 columns of U, 5 rows and 5 columns of S, and first 5 rows of V transpose, and then if you plot this using matrix plot, this is what you get.

(Refer Slide Time: 22:22)



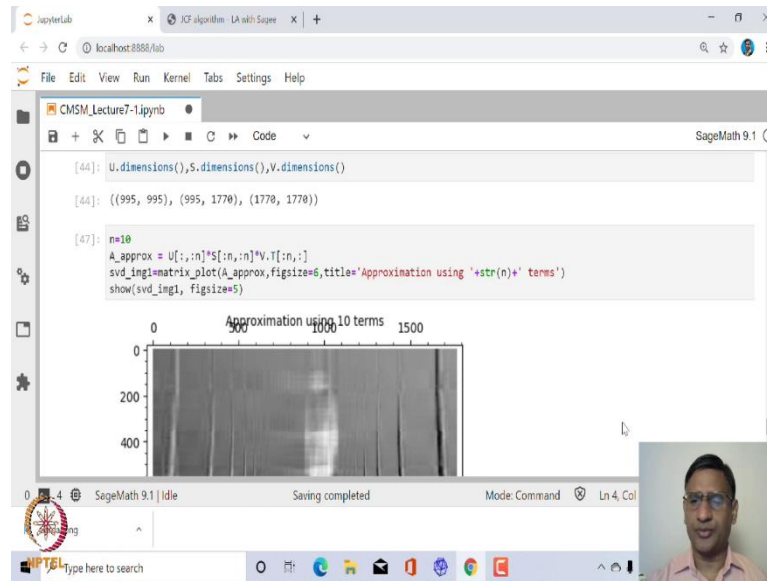
So, you can hardly make out about this image, because this. So, this is not a very good approximation.

(Refer Slide Time: 22:31)



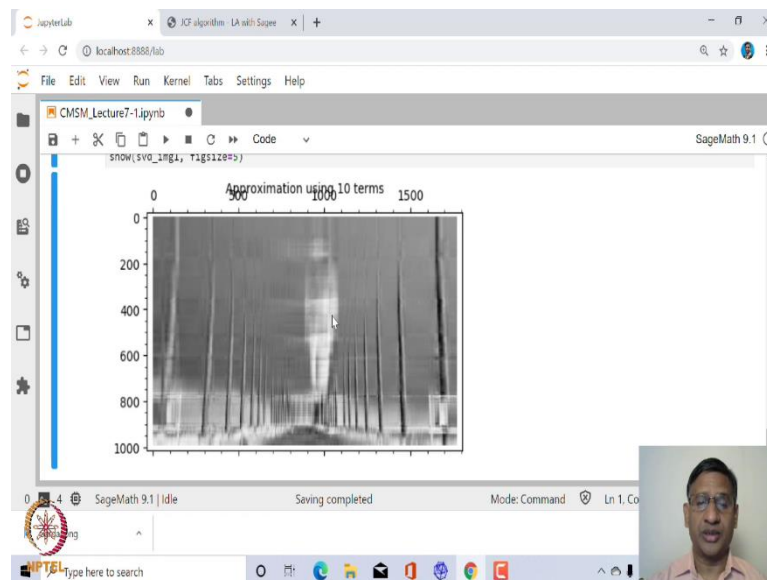
Let me reduce the image size. So, I will say, show this, this and image figure size, fig size equal to, let us say 4, or let me make it 5 yeah. So, this is slightly better. So, this is what you get.

(Refer Slide Time: 22:59)



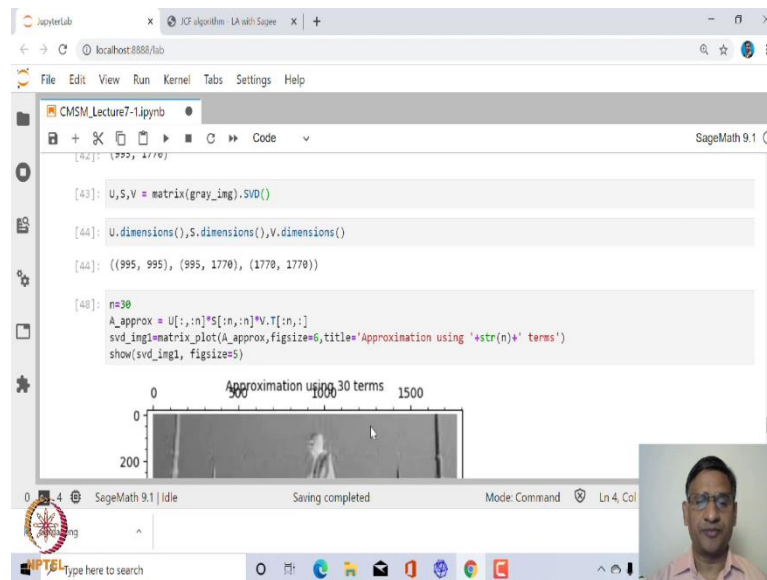
Now, let us increase the number of terms. So, instead of 5, let us make it 10. So, when you make it 10 approximation, now you can see here, this, several things, I mean this is much better than the previous one.

(Refer Slide Time: 23:05)



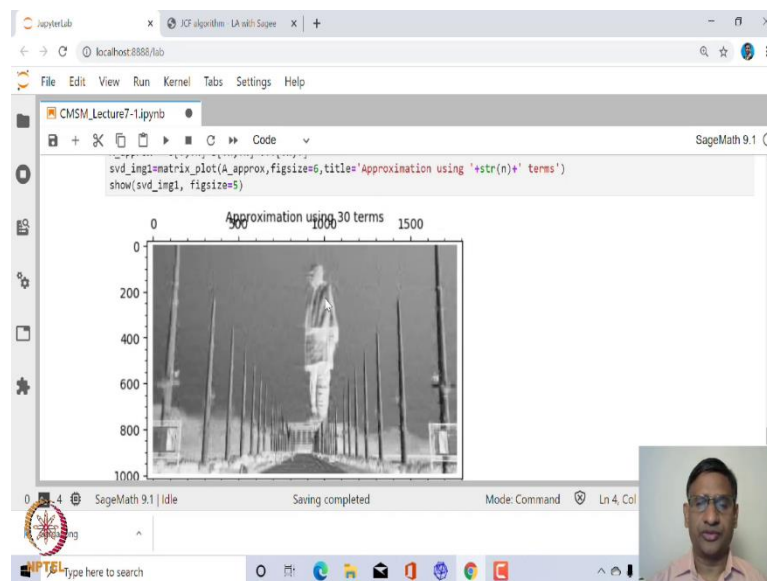
It also looks like that you have several of these iron poles, and this, this is a kind of a statue, right?

(Refer Slide Time: 23:21)



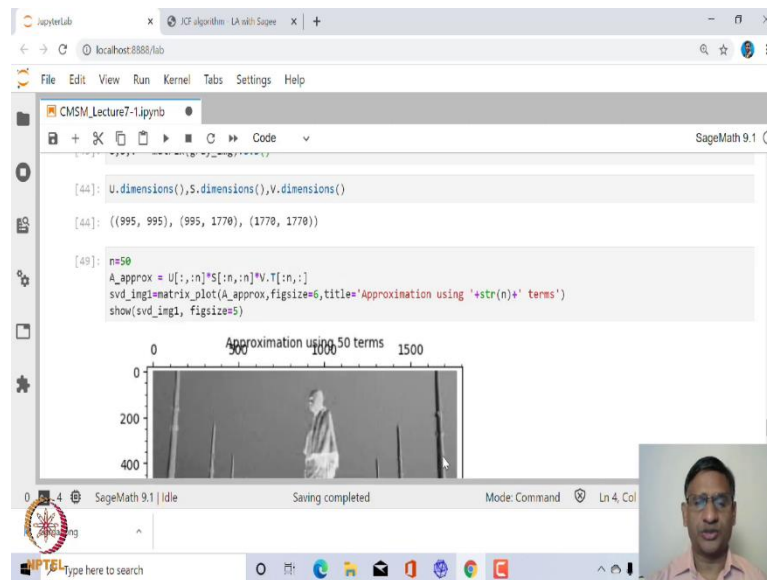
So, instead of 10, if I make it, let us say 30, then it will be much better.

(Refer Slide Time: 23:25)



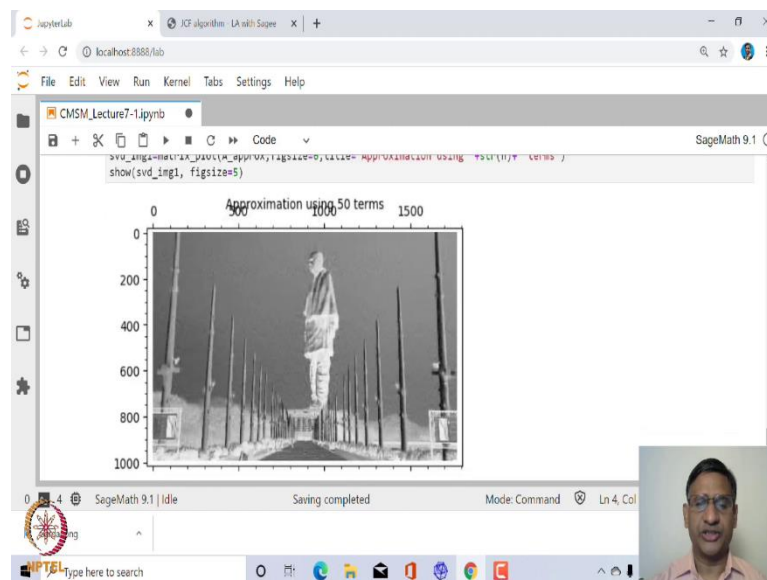
You will be able to see this is almost quite clear.

(Refer Slide Time: 23:32)



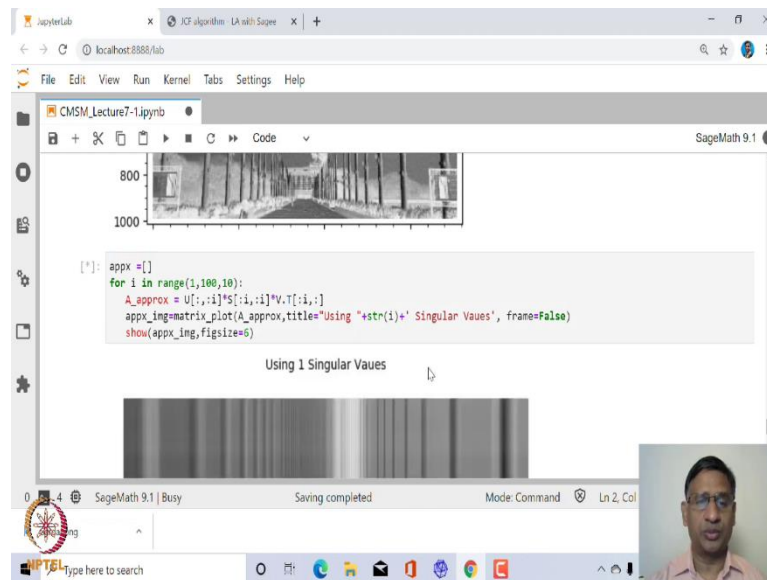
And, if you take about, let us say 100 terms, or let us just take 50 terms, then the approximation is again very much clear, right?

(Refer Slide Time: 23:38)



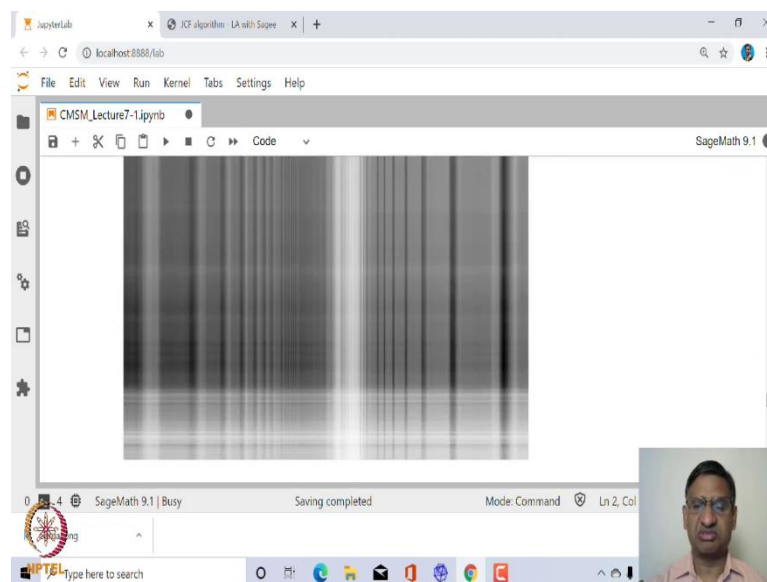
So, you see that first 50 terms itself gives you very nice approximation of the original image, whereas, so first 50 terms in this case, will, if you compare the number of entries in first 50 terms of this approximation, compared to the original matrix, there is substantial difference in this. So, lot of space is saved, right? So, this is what is called matrix, image compression. So, right?

(Refer Slide Time: 24:19)

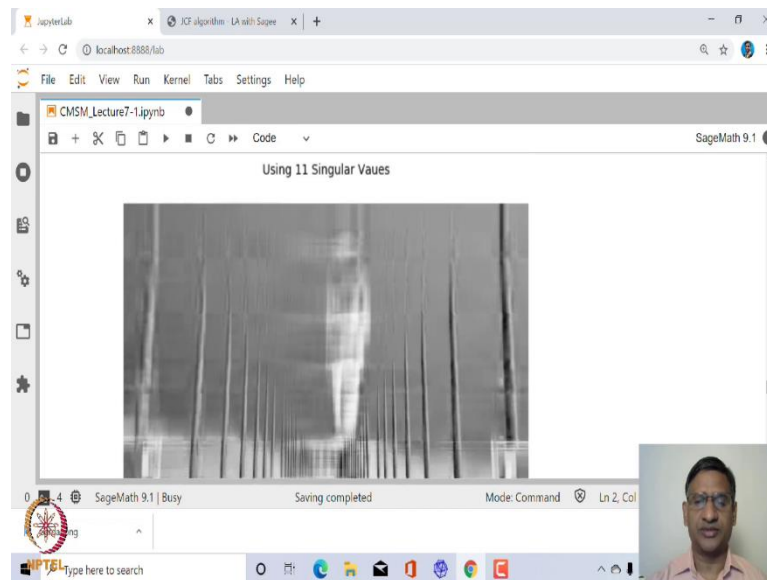


Next let me, let me, let me just run this a loop, so that here it is showing you approximation of, starting with first one 1 term, then 11 term, then 21 terms, and so on. So, let us run this loop.

(Refer Slide Time: 24:35)

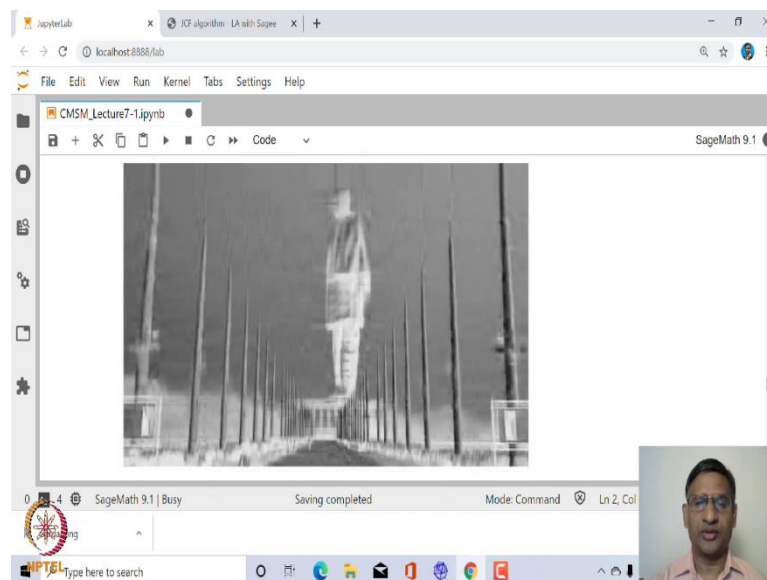


(Refer Slide Time: 24:36)

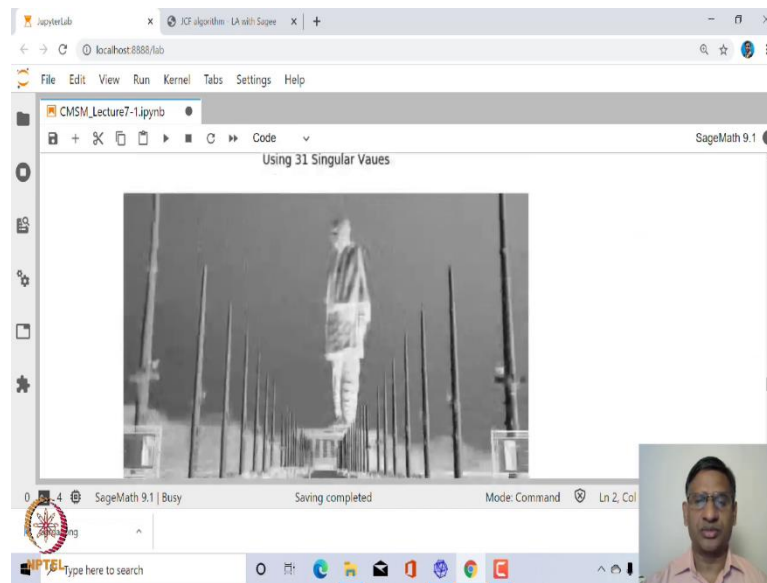


So, first approximation is, you cannot make out anything, this is the first, 11 term. Then 21 term, then 31 terms, 41 terms.

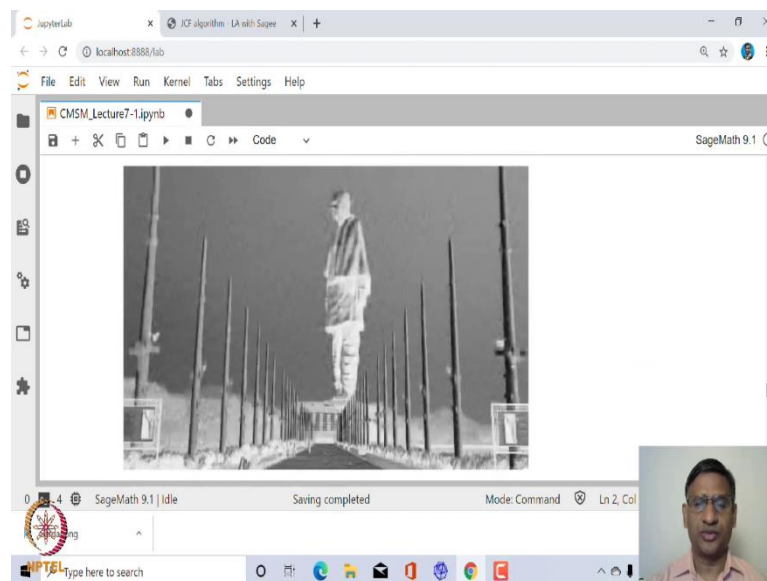
(Refer Slide Time: 24:39)



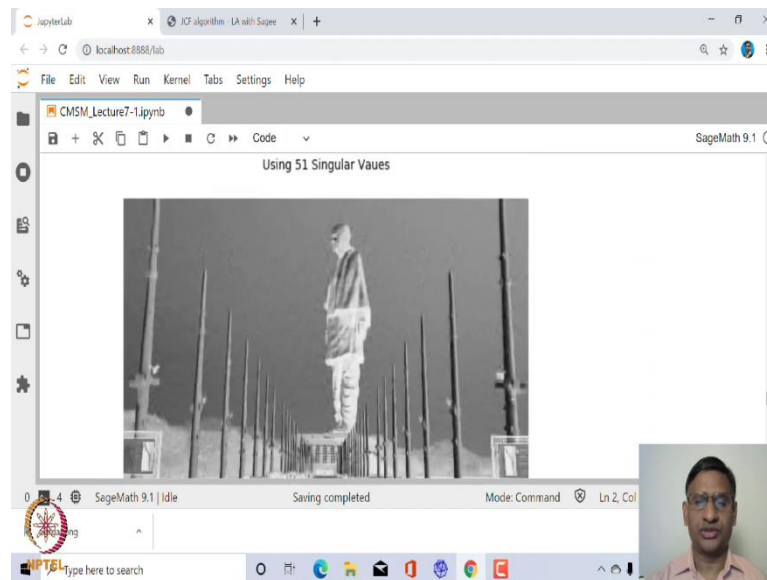
(Refer Slide Time: 24:41)



(Refer Slide Time: 24:43)

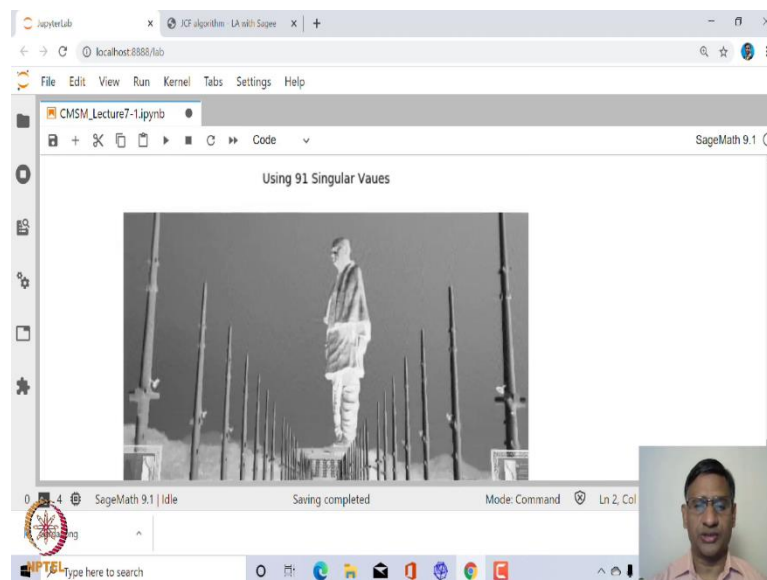


(Refer Slide Time: 24:44)

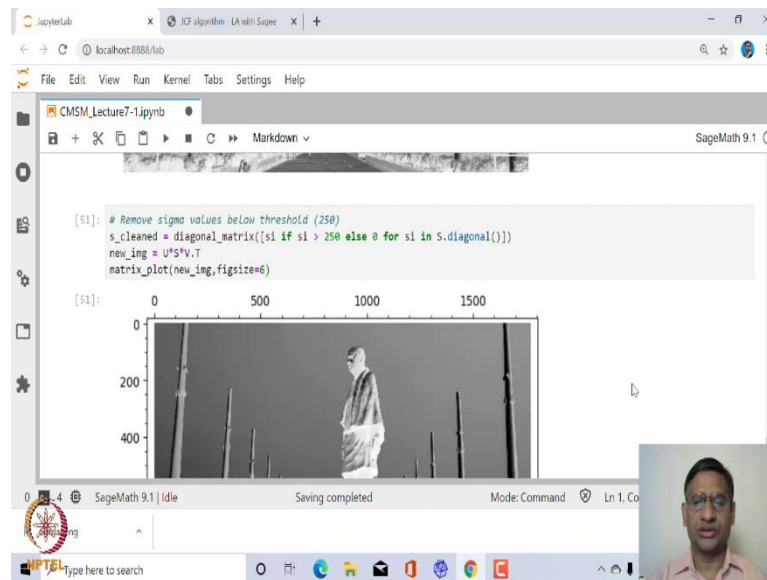


And, you can see here, when you reach about 100, the approximation is very very good, right? It's almost like a original, right?

(Refer Slide Time: 24:50)

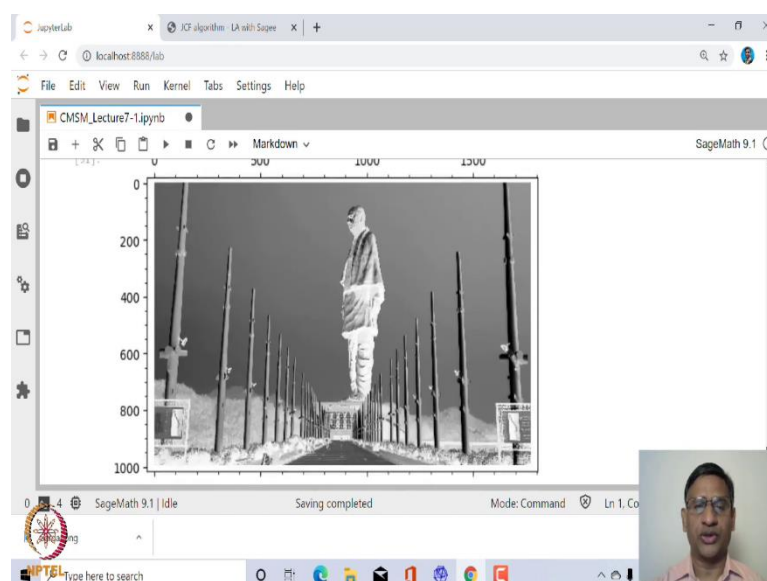


(Refer Slide Time: 24:57)



So, that is what is called image compression, compression and that is an application to SVD. This you can also, just ignore all the diagonal entries, which is, which is less than 250, in, in sigma, and then that is what we call this as denoising this matrix. So, you, you have, have take them, some threshold value and get rid of those diagonal entries and then in this case, if you try to plot this graph, so, this is again quite good compared to the original one.

(Refer Slide Time: 25:34)



So, this is another way of denoising matrix. So, this singular value decomposition had, has several applications. One of them is in image processing, and it has many more applications. For example, in data science, they use a singular value decomposition as dimensionality reduction algorithm. And, in this case, you can see here, instead of taking all that  $r$  terms which is rank of this matrix, we have, we have taken only few, first few terms.

So, you have reduced these several components of this, and still you are getting very good approximation, right? So, let me stop here. We shall look at more applications in, in next class. We will look at application of linear algebra to solving system of linear equations, that will be again an application of eigenvalues, eigenvectors.

Thank you very much.