SINGULAR VALUE DECOMPOSITION(SVD)

Singular Value Decomposition (SVD) is a matrix factorization technique used in linear algebra and numerical analysis. Given a matrix, SVD decomposes it into three matrices: a left singular matrix, a diagonal matrix, and a right singular matrix.

Mathematically, if we have a matrix A of size m × n, its singular value decomposition is expressed as:

$A = USV^T$

Where U is an m × m orthogonal matrix formed by placing the eigenvectors of the matrix $A.A^T$ in the columns of U,
S is an m × n diagonal matrix with non-negative real numbers on the diagonal (known as the singular values). The diagonal entries are the square root of the eigenvalues of the matrix $A^T A$, entered in a decreasing order,
and V^T is an n × n orthogonal matrix formed by putting the eigenvectors of $A^T A$ in the columns of V

Choosing a Matrix

In[ ]:=
```
Clear["Global`*"]
A = {{0,1,1},{0,1,0},{1,1,0},{0,1,0},{1,0,1}};
m = Length[A]; (*This gives the no of rows of the matrix*)
n = Length[Transpose[A]]; (*This gives the no of columns of the matrix*)

MatrixForm[A] (*printing the matrix*)
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

Calculating the Singular Matrix:

In[ ]:=
```
(*calculating the Matrix A^T. A*)
b = Dot[Transpose[A].A];
MatrixForm[b]

Eigenvalues[b]; (*Finding the eigenvakue of the matrix, b*)
singeigb = Sqrt[Eigenvalues[b]]; (*generating the singular values of the matrix*)
S = SparseArray[{{1,1}→singeigb[[1]],
{2,2}→singeigb[[2]],{3,3}→singeigb[[3]]},{m,n}];
MatrixForm[S] (*Printing the Singular Matrix*)
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 2 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

Out[ ]//MatrixForm=

$$\begin{pmatrix} \sqrt{5} & 0 & 0 \\ 0 & \sqrt{2} & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Calculating the V matrix

In[ ]:=
```
eigvectorb = Eigenvectors[b]; (*Finding the eigenvectors of the matrix, b*)

(*Normalizing the eigenvectors and putting them in the columns of the vector, V*)
V =  Transpose[Map[Normalize,eigvectorb]];
MatrixForm@V
transV = Transpose[V];
```

Out[ ]//MatrixForm=

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

Calculating the U matrix

In this case we first find out the eigenvectors of AA^T. We observed that there are five eigenvectors, namely x1,x2,x3,x4,x5. It is noticed that either x4 or x5 is orthogonal to the rest of the eigenvectors. So we applied the Gram Schmidt orthogonalization to the vectors x4 and x5 and then normalized all the sets of eigenvectors.

```
c = Dot[A.Transpose[A]]; (*calculating the Matrix, A.A^T*)
MatrixForm@c  (*printing the A.A^T Matrix*)
eigvc = Eigenvectors[c];
MatrixForm/@eigvc   (*Printing the eigenvectors of c *)

(*Applying the Gram Schmidt Orthogonalization on the eigenvectors, x4 and x5*)
v5 = eigvc[[5]] - Dot[Transpose[eigvc[[4]].eigvc[[5]]] / Norm[eigvc[[4]]]^2 *eigvc[[4]];

(*Replacing the non-othogonal eigenvector with the orthogonal one*)
orthoeigvecc = ReplacePart[eigvc,5→v5];
MatrixForm/@orthoeigvecc  (*Printing the orthonormal eigenvectors*)

(*Storing the normalized eigenvectors into the columns of the Matrix,U*)
U = Transpose[Map[Normalize,orthoeigvecc]];

MatrixForm@U  (*Printing the U Matrix*)

A = U.S.Transpose[V];
MatrixForm[A];
Print["A = U.S.Transpose[V]:\n", MatrixForm[A]]
   (*Verifying the Singular Value Decomposition*)
```

Out[•]//MatrixForm=

$$\begin{pmatrix} 2 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 2 \end{pmatrix}$$

Out[•]=

$$\left\{\begin{pmatrix} 3 \\ 2 \\ 3 \\ 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ 0 \\ -1 \\ 2 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 2 \\ -1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ 0 \\ 1 \\ 0 \end{pmatrix}\right\}$$

Out[•]=

$$\left\{\begin{pmatrix} 3 \\ 2 \\ 3 \\ 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ 0 \\ -1 \\ 2 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 2 \\ -1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -\frac{2}{7} \\ -\frac{3}{7} \\ -\frac{2}{7} \\ 1 \\ \frac{2}{7} \end{pmatrix}\right\}$$

Out[•]//MatrixForm=

$$\begin{pmatrix} \sqrt{\frac{3}{10}} & 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{7}} & -\sqrt{\frac{2}{35}} \\ \sqrt{\frac{2}{15}} & -\frac{1}{\sqrt{6}} & 0 & \frac{2}{\sqrt{7}} & -\frac{3}{\sqrt{70}} \\ \sqrt{\frac{3}{10}} & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{7}} & -\sqrt{\frac{2}{35}} \\ \sqrt{\frac{2}{15}} & -\frac{1}{\sqrt{6}} & 0 & 0 & \sqrt{\frac{7}{10}} \\ \sqrt{\frac{2}{15}} & \sqrt{\frac{2}{3}} & 0 & \frac{1}{\sqrt{7}} & \sqrt{\frac{2}{35}} \end{pmatrix}$$

```
A = U.S.Transpose[V]:
```

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

Another Method of finding U

In this method we find the column vectors of U by using the formula $\mathbf{U_i = \frac{1}{s_i} A.V_i^T}$. We see that for the non-zero eigenvalues we can easily find the column vectors. To find the rest of the vectors we choose any two orthogonal vectors and normalized them and formed the complete U matrix.

```
(*Here we used the formula U_i = 1/s_i A.V_i^T
We can calculate the eigenvecotors corresponding to the Non-zero Singular values of S*)
eiglist = Table[Dot[1/singeigb[[i]] * (A.Transpose[transV[[i]]])],{i,1,3}]//FullSimplify;
(*To find other two eigenvectors we choose two orthogonal vectors*)
list2 = {{1,-1,-1,1,-1},{1,1,0,1,1}};
list3  = Flatten[{eiglist,Map[Normalize,list2]},1];  (*Normalizing the eigenvectors*)
MatrixForm/@list3  (*Printing the orthonormal eigenvectors*)
U1 = Transpose[list3];
MatrixForm@U1 (*Showing the Matrix U*)
Dot[U1.S.Transpose[V]]//MatrixForm   (*Verifying the Singular Value Decomposition*)
```

Out[•]=

$$\left\{\begin{pmatrix}\sqrt{\frac{3}{10}}\\ \sqrt{\frac{2}{15}}\\ \sqrt{\frac{3}{10}}\\ \sqrt{\frac{2}{15}}\\ \sqrt{\frac{2}{15}}\end{pmatrix},\begin{pmatrix}0\\ -\frac{1}{\sqrt{6}}\\ 0\\ -\frac{1}{\sqrt{6}}\\ \sqrt{\frac{2}{3}}\end{pmatrix},\begin{pmatrix}-\frac{1}{\sqrt{2}}\\ 0\\ \frac{1}{\sqrt{2}}\\ 0\\ 0\end{pmatrix},\begin{pmatrix}\frac{1}{\sqrt{5}}\\ -\frac{1}{\sqrt{5}}\\ -\frac{1}{\sqrt{5}}\\ \frac{1}{\sqrt{5}}\\ -\frac{1}{\sqrt{5}}\end{pmatrix},\begin{pmatrix}\frac{1}{2}\\ \frac{1}{2}\\ 0\\ \frac{1}{2}\\ \frac{1}{2}\end{pmatrix}\right\}$$

Out[•]//MatrixForm=

$$\begin{pmatrix}\sqrt{\frac{3}{10}} & 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{5}} & \frac{1}{2}\\ \sqrt{\frac{2}{15}} & -\frac{1}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{5}} & \frac{1}{2}\\ \sqrt{\frac{3}{10}} & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{5}} & 0\\ \sqrt{\frac{2}{15}} & -\frac{1}{\sqrt{6}} & 0 & \frac{1}{\sqrt{5}} & \frac{1}{2}\\ \sqrt{\frac{2}{15}} & \sqrt{\frac{2}{3}} & 0 & -\frac{1}{\sqrt{5}} & \frac{1}{2}\end{pmatrix}$$

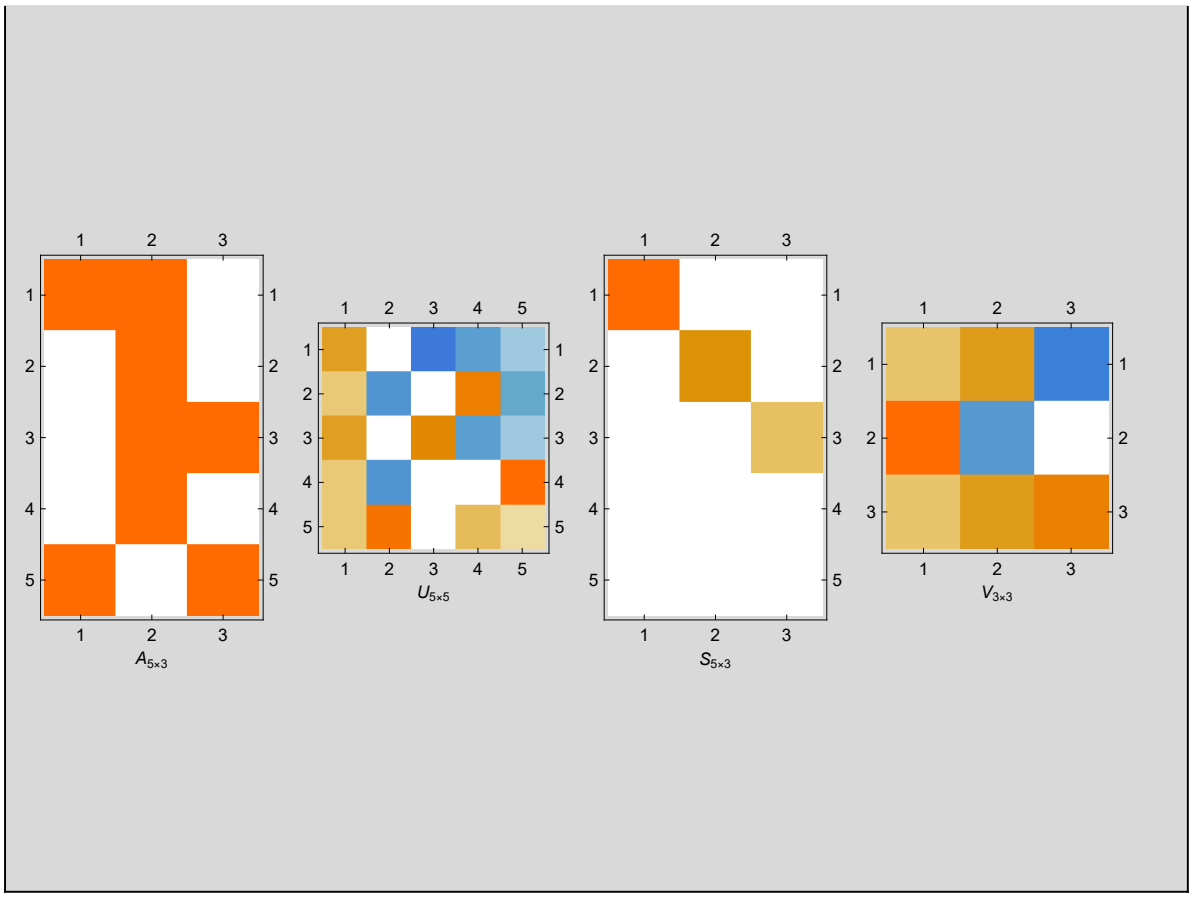
Out[•]//MatrixForm=

$$\begin{pmatrix}1 & 1 & 0\\ 0 & 1 & 0\\ 0 & 1 & 1\\ 0 & 1 & 0\\ 1 & 0 & 1\end{pmatrix}$$

Visual Representation of SVD

In[•]:=
```
GraphicsGrid[{
  {Labeled[MatrixPlot[A], "A5×3"], Labeled[MatrixPlot[U], "U5×5"],
  Labeled[MatrixPlot[S], "S5×3"], Labeled[MatrixPlot[V], "V3×3"]}}, AspectRatio→1, ImageSize→L
```

Out[•]=



Trying with another example

Here we take the matrix: A = {{1,-1},{1,1},{0,1},{1,0},{-1,1}}

In[•]:=
```
Clear["Global`*"]
A = {{1,-1},{1,1},{0,1},{1,0},{-1,1}};
m = Length[A];
n = Length[Transpose[A]];
MatrixForm[A] (*Printing the A matrix*)

b = Dot[Transpose[A].A];
MatrixForm[b]

singeigb = Sqrt[Eigenvalues[b]];
S =  SparseArray[{{1,1}→singeigb[[1]],
{2,2}→singeigb[[2]]},{m,n}];
MatrixForm[S] (*Printing the Singular Matrix*)


(*Calculating the V matrix*)
eigvectorb = Eigenvectors[b];
V =  Transpose[Map[Normalize,eigvectorb]];
MatrixForm@V
transV = Transpose[V];
```

Out[•]//MatrixForm=

$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 1 \end{pmatrix}$$

Out[•]//MatrixForm=

$$\begin{pmatrix} 4 & -1 \\ -1 & 4 \end{pmatrix}$$

Out[•]//MatrixForm=

$$\begin{pmatrix} \sqrt{5} & 0 \\ 0 & \sqrt{3} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Out[•]//MatrixForm=

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

Calculating the U Matrix

In[•]:=
```
c = Dot[A.Transpose[A]];
MatrixForm@c
eigvc = Eigenvectors[c];
MatrixForm/@eigvc    (*Printing the eigenvectors of the Matrix, A.A^T*)
vecset = eigvc[[3;;5]];

(*Applying the Gram-Schmidth Orthogonalization on the last 3 eigenvectors
*)
vectors = vecset;

(* Applying the Gram-Schmidt process to the set of vectors *)
n = Length[vectors];
u = ConstantArray[0, {n, n}];
u[[1]] = vectors[[1]]/Norm[vectors[[1]]];
For[i = 2, i ≤ n, i++,
  ui = vectors[[i]];
  For[j = 1, j < i, j++,
    uj = u[[j]];
    ui = ui - (uj.ui/uj.uj) uj;
  ];
  u[[i]] = ui;
  u[[i]] = u[[i]] / Sqrt[u[[i]].u[[i]]];
]

(* Printing the resulting set of normalized vectors *)
 normalu = u//FullSimplify;
MatrixForm/@u//FullSimplify;

(*Printing the normal eigenvectors*)
normaleigu = Flatten[{Map[Normalize,eigvc[[1;;2]]],normalu},1];
MatrixForm/@normaleigu

U  = Transpose[normaleigu];
MatrixForm[U] (*Printing the Matrix U*)
Dot[U.S.Transpose[V]]//MatrixForm    (*Verifying the SVD*)
```

Out[•]//MatrixForm=

$$\begin{pmatrix} 2 & 0 & -1 & 1 & -2 \\ 0 & 2 & 1 & 1 & 0 \\ -1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & -1 \\ -2 & 0 & 1 & -1 & 2 \end{pmatrix}$$

Out[•]=

$$\left\{\begin{pmatrix} -2 \\ 0 \\ 1 \\ -1 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \\ 0 \\ 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 2 \\ 0 \\ 0 \end{pmatrix}\right\}$$

Out[•]=

$$\left\{\begin{pmatrix} -\sqrt{\frac{2}{5}} \\ 0 \\ \frac{1}{\sqrt{10}} \\ -\frac{1}{\sqrt{10}} \\ \sqrt{\frac{2}{5}} \end{pmatrix}, \begin{pmatrix} 0 \\ \sqrt{\frac{2}{3}} \\ \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \\ 0 \end{pmatrix}, \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}, \begin{pmatrix} -\frac{1}{\sqrt{22}} \\ -\sqrt{\frac{2}{11}} \\ 0 \\ 2\sqrt{\frac{2}{11}} \\ \frac{1}{\sqrt{22}} \end{pmatrix}, \begin{pmatrix} \sqrt{\frac{3}{55}} \\ -\sqrt{\frac{5}{33}} \\ \sqrt{\frac{11}{15}} \\ -\frac{1}{\sqrt{165}} \\ -\sqrt{\frac{3}{55}} \end{pmatrix}\right\}$$

Out[•]//MatrixForm=

$$\begin{pmatrix} -\sqrt{\frac{2}{5}} & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{22}} & \sqrt{\frac{3}{55}} \\ 0 & \sqrt{\frac{2}{3}} & 0 & -\sqrt{\frac{2}{11}} & -\sqrt{\frac{5}{33}} \\ \frac{1}{\sqrt{10}} & \frac{1}{\sqrt{6}} & 0 & 0 & \sqrt{\frac{11}{15}} \\ -\frac{1}{\sqrt{10}} & \frac{1}{\sqrt{6}} & 0 & 2\sqrt{\frac{2}{11}} & -\frac{1}{\sqrt{165}} \\ \sqrt{\frac{2}{5}} & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{22}} & -\sqrt{\frac{3}{55}} \end{pmatrix}$$

Out[•]//MatrixForm=

$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 1 \end{pmatrix}$$

Visual Representation of the SVD

In[•]:=
```
GraphicsGrid[{
  {Labeled[MatrixPlot[A], "A5×2"], Labeled[MatrixPlot[U], "U5×5"],
  Labeled[MatrixPlot[S], "S5×2"], Labeled[MatrixPlot[V], "V2×2"]}}, AspectRatio→1, ImageSize→L
```

Out[•]=

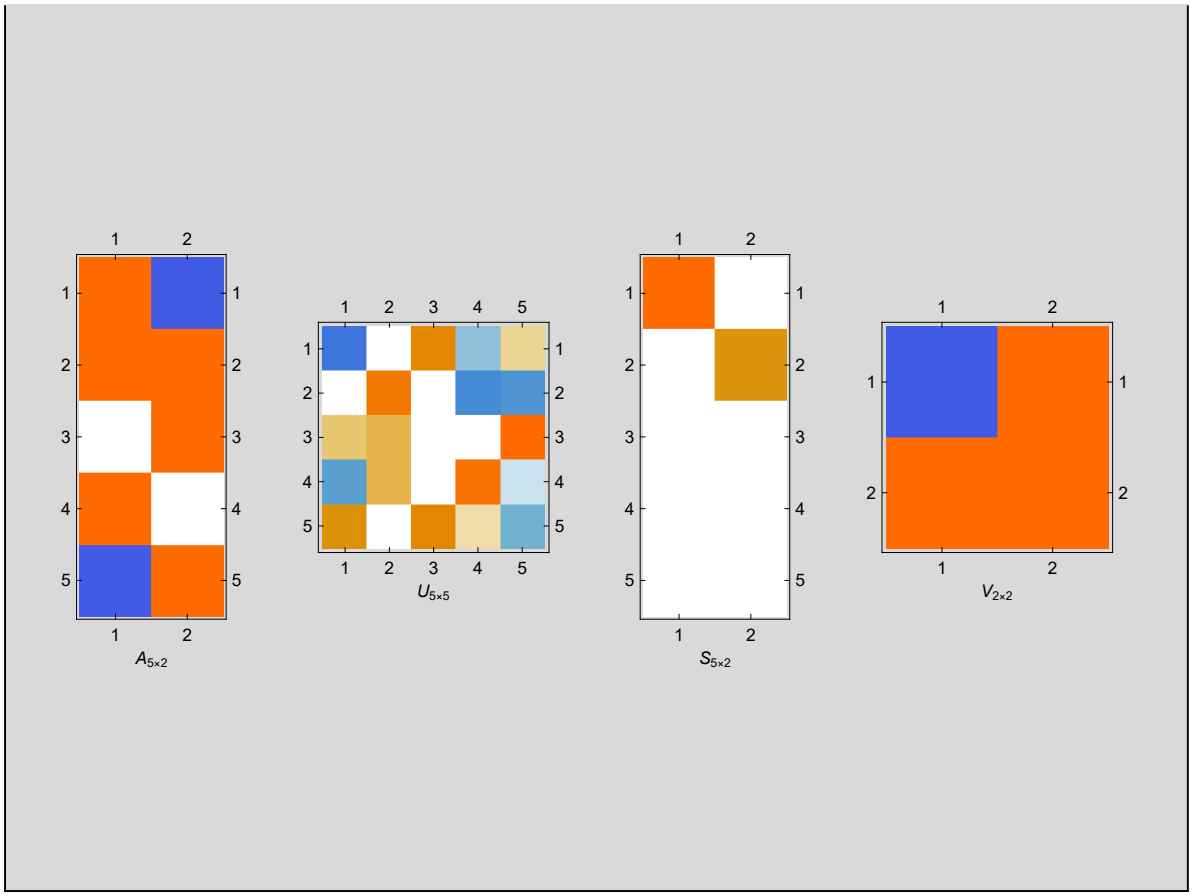


IMAGE COMPRESSION : AN APPLICATION OF SVD

SVD can be used to compress an image by representing the image in terms of a smaller number of singular values and vectors. The idea is to keep only the most important singular values and vectors, and discard the rest. This results in a compressed representation of the image that takes up less space, while still preserving much of the original image information.
Here's a general overview of the steps involved in using SVD for image compression:

1. **Convert the image into a matrix**: To use SVD, we need to represent the image as a matrix. We can do this by assigning a numerical value to each pixel of the image, and arranging these values into a matrix. The size of the matrix will depend on the resolution of the image.

2. **Perform SVD on the matrix:** Once we have the matrix representation of the image, we can perform SVD on it to obtain the left singular matrix (U), the diagonal matrix (Σ), and the right singular matrix (V^T).

3. **Choose the number of singular values to keep:** We can choose to keep only a subset of the singular values and vectors, based on the amount of compression we want to achieve. Typically, we would keep the singular values with the largest magnitudes, as they represent the most important information about the image.

4. **Reconstruct the image using the truncated SVD**: To compress the image, we can truncate the SVD matrices by keeping only the selected singular values and vectors, and discarding the rest. We can then reconstruct the compressed image by multiplying the truncated matrices back together.

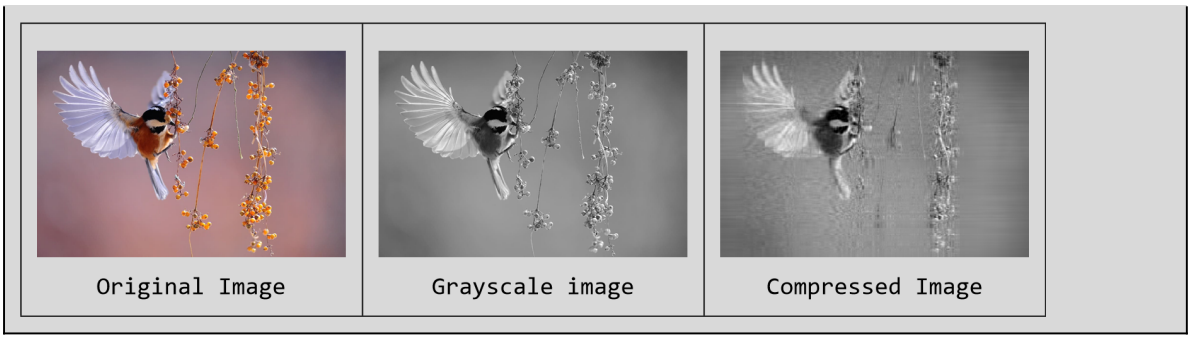
5. **Compare the compressed and original images:** We can compare the compressed image to the original image to assess the level of compression achieved. Typically, there will be some loss of image quality, but the amount of loss can be controlled by adjusting the number of singular values kept.

Overall, SVD-based image compression can be a useful technique for reducing the storage and transmission requirements of image data, while still preserving much of the original image information.

In[246]:=

```
Clear["Global`*"]
(* Loading an image *)
img = Import["C:\\Users\\HP\\Downloads\\trial image.jpg"];

(* Converting the image into grayscale *)
imgGray = ColorConvert[img, "Grayscale"];

(* Performing singular value decomposition (SVD) *)
{U, S, V} = SingularValueDecomposition[ImageData[imgGray]];

(* Specifying the number of singular values to keep for compression *)
truncatedsingularvalues = 25;

(* Extracting the diagonal of the S matrix *)
sValues = Diagonal[S];

(* Truncating the singular value vectors to keep only a specified number of singular values *)
sValuesTruncated = Take[sValues, truncatedsingularvalues];
UTruncated = Take[U, All, truncatedsingularvalues];
VTruncated = Take[V, All, truncatedsingularvalues];

(* Reconstructing the compressed image *)
compressedImg = Image[UTruncated.DiagonalMatrix[sValuesTruncated].Transpose[VTruncated]];

(* Displaying the original and compressed images *)
Grid[{{Magnify[Labeled[img,"Original Image"],1.5] ,
 Magnify[Labeled[imgGray,"Grayscale image"],1.5],
Magnify[Labeled[compressedImg,"Compressed Image"],1.5]}}, Dividers→All, Spacings→{2, 2}]
```

Out[•]=



Calculating the Compression ratio and percentage of Compression

In[49]:=

```
(* Calculating the compression ratio *)
compressionRatio = N[(truncatedsingularvalues)/(Length[sValues])];

(* Calculating  percentage compression *)
percentagecompression = 100.0 (1 - (truncatedsingularvalues/Length[sValues]));
Print["The compression ratio is ", compressionRatio]
Print["The percentage of compression is ", percentagecompression, "%"]
```

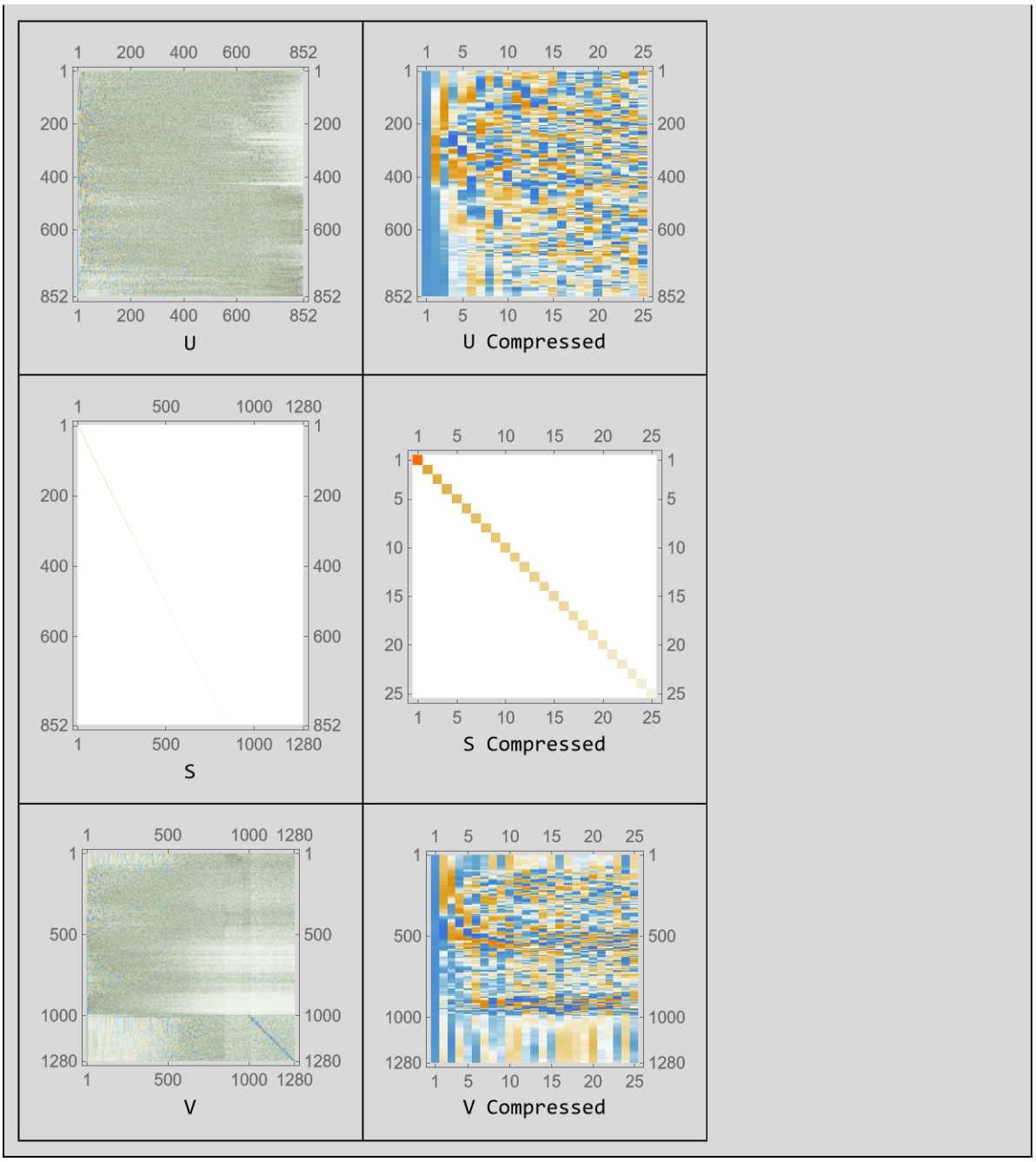
```
The compression ratio is 0.0293427

The percentage of compression is 97.0657%
```

MatrixPlot of the Decomposed Matrices, U,S,V

In[257]:=

```
Grid[{{Labeled[MatrixPlot[U],"U"], Labeled[MatrixPlot[UTruncated, AspectRatio→1],"U Compresse
{Labeled[MatrixPlot[S],"S"],
Labeled[MatrixPlot[DiagonalMatrix[sValuesTruncated]],"S Compressed"]},
{Labeled[MatrixPlot[V],"V"],Labeled[MatrixPlot[VTruncated, AspectRatio→1],"V Compressed"]}},
Dividers→All, Spacings→{2, 2}]
```

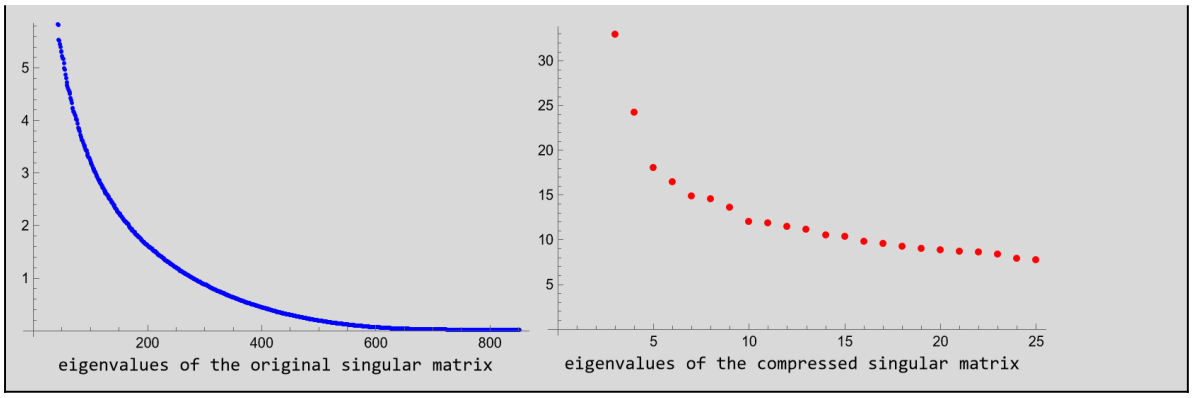
Out[•]=



Plotting the eigenvalues of the Original and Truncated singular matrices.

```
Grid[{{Labeled[ListPlot[Diagonal[S], ImageSize→Medium, PlotStyle→Blue],
"eigenvalues of the original singular matrix"],
Labeled[ListPlot[sValuesTruncated, ImageSize→Medium, PlotStyle→Red],
"eigenvalues of the compressed singular matrix"]}}]
```

Out[•]=



Compressing the original image instead of converting it into GrayScale

To comperes the image in the original color the following steps are followed:

1. **Convert the image into a matrix:** As with grayscale images, we need to represent the color image as a matrix to perform SVD. However, for color images, we first need to decompose the image into its Red, Green, and Blue channels. This is done by separating the intensity values of each pixel into three separate matrices, one for each color channel. Each of these matrices will have the same dimensions as the original image.

2. **Perform SVD on each color channel:** Once we have the separate matrices for each color channel, we can perform SVD on each one individually to obtain the left singular matrix (U), the diagonal matrix (Σ), and the right singular matrix (V^T).

3. **Choose the number of singular values to keep for each color channel**: As with grayscale images, we can choose to keep only a subset of the singular values and vectors for each color channel. Typically, we would keep the singular values with the largest magnitudes, as they represent the most important information about the image.

4. **Truncate the singular values for each color channel**: We can truncate the SVD matrices for each color channel by keeping only the selected singular values and vectors, and discarding the rest. This results in compressed matrices for each color channel.

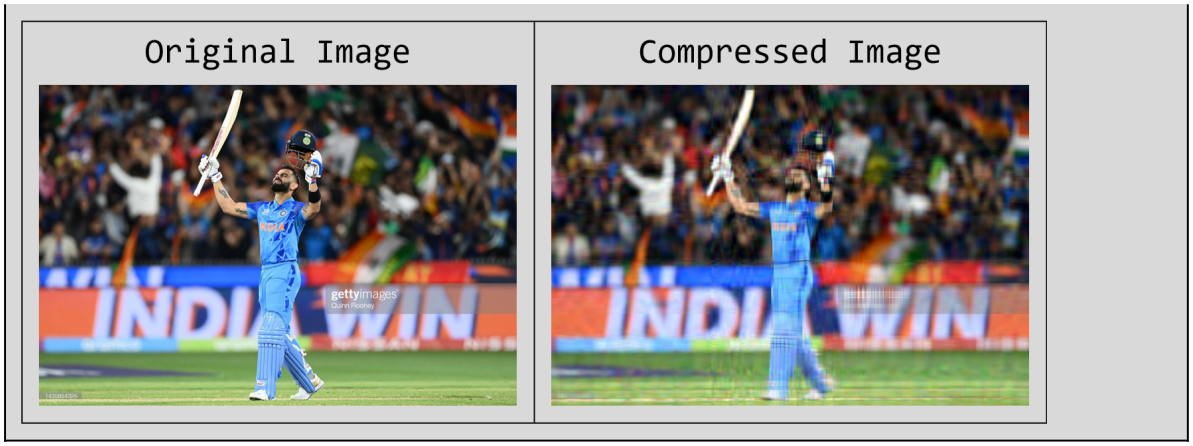
5. **Reconstruct the truncated pixel matrix for each color channel:** To compress the image while preserving its original color, we need to reconstruct the compressed matrices for each color channel. This is done by multiplying the truncated matrices back together. This results in a compressed pixel matrix for each color channel.

6. **Recombine the RGB channels**: Finally, we can recombine the compressed pixel matrices for each color channel into a single compressed image matrix, and convert it back into an image format. This will result in a compressed color image that is similar to the original image, but with some loss of information due to the compression.

In[259]:=

```
Clear["Global`*"]
(* Loading an image *)
img = Import["C:\\Users\\HP\\Downloads\\vk.jpg"];

(* Extracting the RGB channels *)
{rChannel, gChannel, bChannel} = ColorSeparate[img];

(* Performing SVD on each channel separately *)
{Ur, Sr, Vr} = SingularValueDecomposition[ImageData[rChannel]];
{Ug, Sg, Vg} = SingularValueDecomposition[ImageData[gChannel]];
{Ub, Sb, Vb} = SingularValueDecomposition[ImageData[bChannel]];

(* Specifying the number of singular values to keep for compression *)
truncatedsingularvalues = 25;

(* Truncating the singular value matrices for each color channel *)
sValuesTruncatedR = Take[Diagonal[Sr], truncatedsingularvalues];
sValuesTruncatedG = Take[Diagonal[Sg], truncatedsingularvalues];
sValuesTruncatedB = Take[Diagonal[Sb], truncatedsingularvalues];

(* Truncating the U matrices for each color channel *)
UReducedR = Take[Ur, All, truncatedsingularvalues];
UReducedG = Take[Ug, All, truncatedsingularvalues];
UReducedB = Take[Ub, All, truncatedsingularvalues];

(* Truncating the V matrices for each color channel *)
VReducedR = Take[Vr, All, truncatedsingularvalues];
VReducedG = Take[Vg, All, truncatedsingularvalues];
VReducedB = Take[Vb, All, truncatedsingularvalues];

(* Reconstructing the compressed image for each channel *)
compressedR = Image[UReducedR . DiagonalMatrix[sValuesTruncatedR] . Transpose[VReducedR]];
compressedG = Image[UReducedG . DiagonalMatrix[sValuesTruncatedG] . Transpose[VReducedG]];
compressedB = Image[UReducedB . DiagonalMatrix[sValuesTruncatedB] . Transpose[VReducedB]];

(* Combining the compressed RGB channels into a single image *)
compressedImg = Image[Transpose[{ImageData[compressedR],
ImageData[compressedG], ImageData[compressedB]}, {3, 1, 2}]];

(* Displaying the original and compressed images *)
Grid[{{Magnify[Labeled[img, "Original Image", Top], 2],
       Magnify[Labeled[compressedImg, "Compressed Image", Top], 2]}},
  Dividers → All, Spacings → {2, 2}]
```

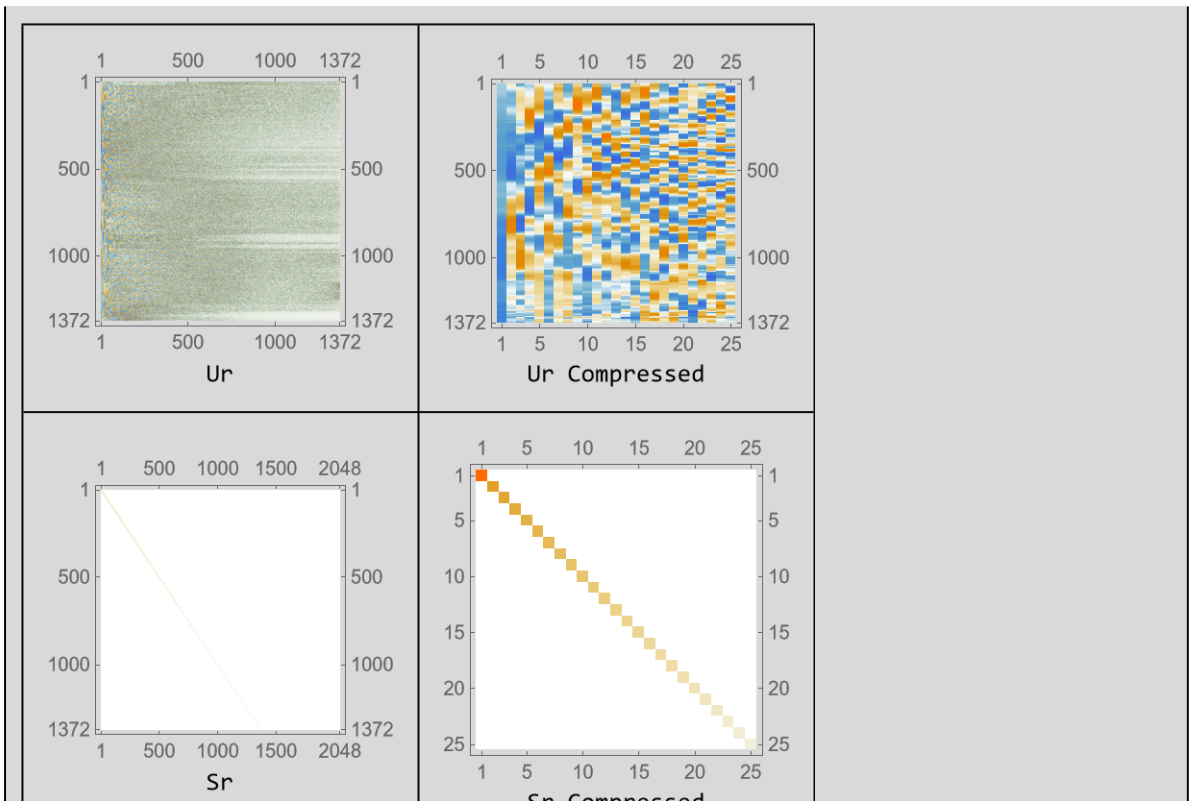
Out[•]=

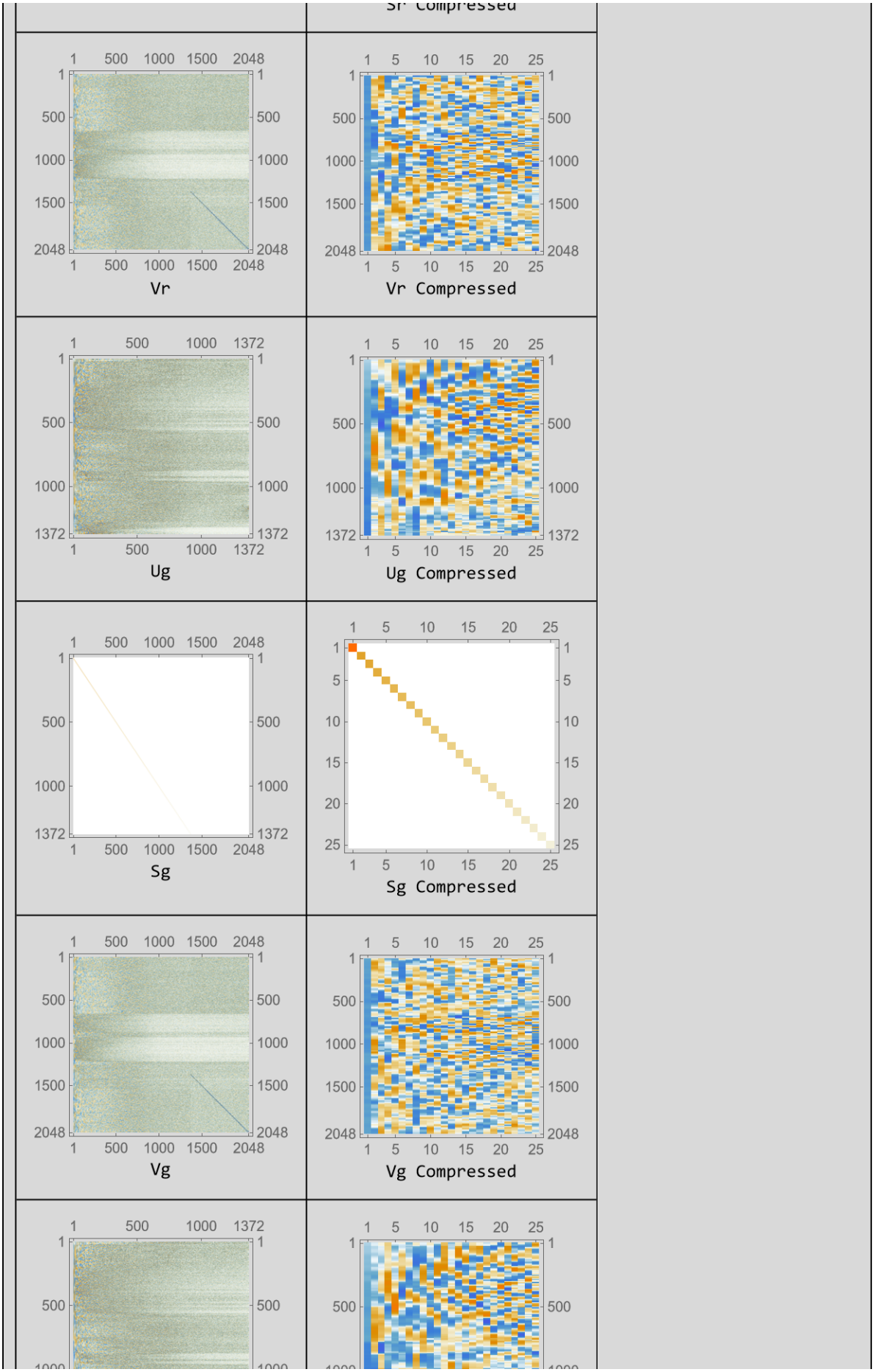


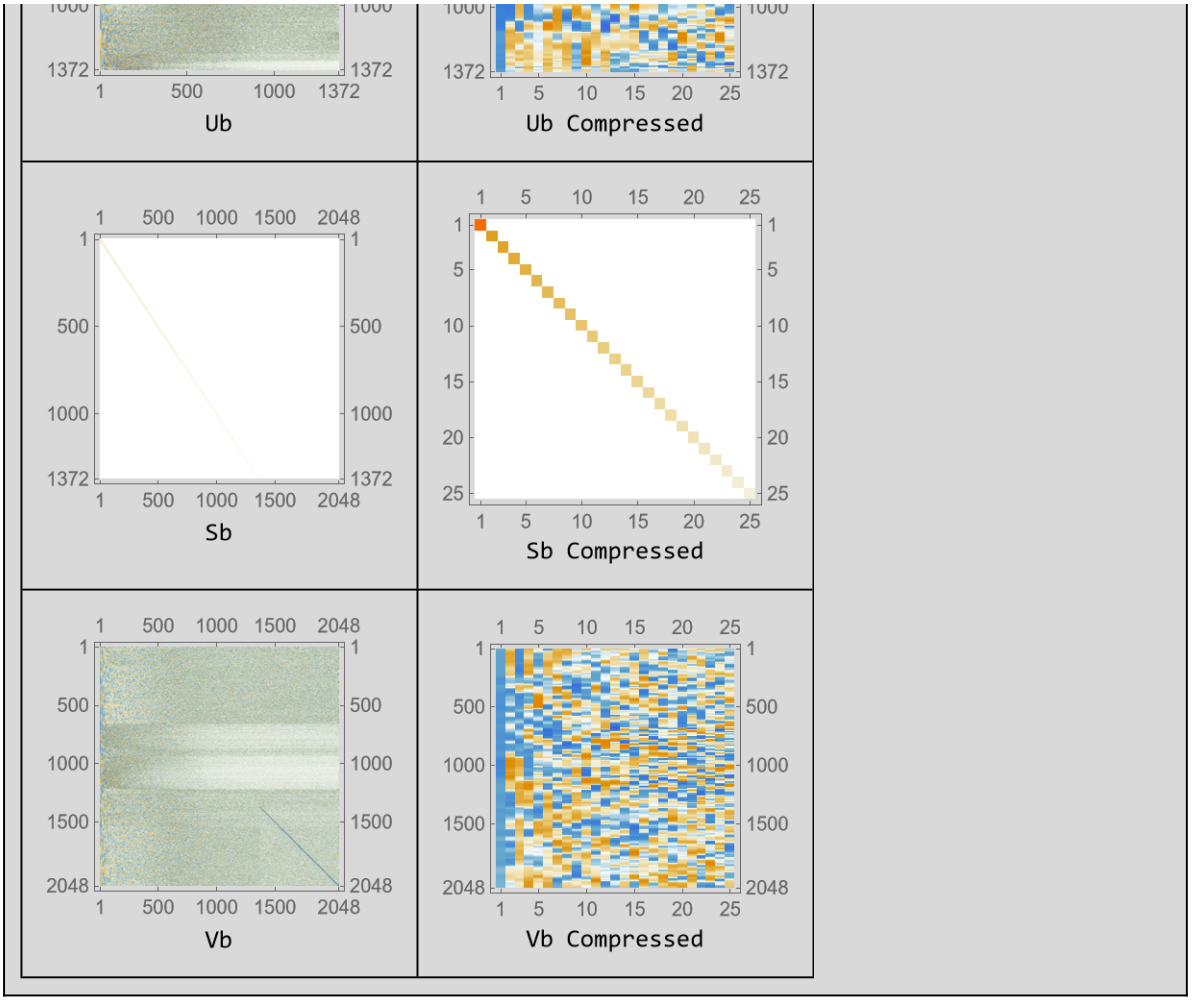
In[282]:=

```
Grid[{
  {Labeled[MatrixPlot[Ur], "Ur"], Labeled[MatrixPlot[UReducedR, AspectRatio → 1], "Ur Compres
  {Labeled[MatrixPlot[Sr], "Sr"], Labeled[MatrixPlot[DiagonalMatrix[sValuesTruncatedR]], "Sr
  {Labeled[MatrixPlot[Vr], "Vr"], Labeled[MatrixPlot[VReducedR, AspectRatio → 1], "Vr Compres

  {Labeled[MatrixPlot[Ug], "Ug"], Labeled[MatrixPlot[UReducedG, AspectRatio → 1], "Ug Compres
  {Labeled[MatrixPlot[Sg], "Sg"], Labeled[MatrixPlot[DiagonalMatrix[sValuesTruncatedG]], "Sg
  {Labeled[MatrixPlot[Vg], "Vg"], Labeled[MatrixPlot[VReducedG, AspectRatio → 1], "Vg Compres

  {Labeled[MatrixPlot[Ub], "Ub"], Labeled[MatrixPlot[UReducedB, AspectRatio → 1], "Ub Compres
  {Labeled[MatrixPlot[Sb], "Sb"], Labeled[MatrixPlot[DiagonalMatrix[sValuesTruncatedB]], "Sb
  {Labeled[MatrixPlot[Vb], "Vb"], Labeled[MatrixPlot[VReducedB, AspectRatio → 1], "Vb Compres
}, Dividers → All, Spacings → {2, 2}]
```

Out[•]=




Vr
Vr Compressed
Ug
Ug Compressed
Sg
Sg Compressed
Vg
Vg Compressed



Reference: The above figure has been taken from, Hyperlink[“Getty Images”, “https://www.gettyimages.in/”].
Link to the picture: “https://www.gettyimages.in/detail/news-photo/virat-kohli-of-india-celebrates-winning-the-icc-mens-t20-news-photo/1435854325?adppopup=true”.

Compressing the image for different Truncated values

```
Clear["Gloabl`*"]
    (* Loading an image *)
    img = Import["C:\\Users\\HP\\OneDrive\\Pictures\\Saved Pictures\\audi_r8_v10_side_view_ro

    (* Converting the image to grayscale *)
    imgGray = ColorConvert[img, "Grayscale"];

    (* Performing singular value decomposition (SVD) *)
    {U, S, V} = SingularValueDecomposition[ImageData[imgGray]];

    (* Specifing the list of singular values to keep for compression *)
    truncationlimitlist = Range[10, 140, 20];

    (* Creating an empty list to store the compressed images *)
    compressedImages = {};
```

```
    (* Creating labels for the compressed images *)
    labels = {};
    comratio = {};
    comper = {};
    (* Iterating over the list of singular values *)
    Do[
      (* Extracting the diagonal of the S matrix *)
      sValues = Diagonal[S];

      (* Truncating the singular value vectors to keep only the specified
      number of singular values *)
      sValuesTruncated = Take[sValues, truncationlimit];
      UTruncated = Take[U, All, truncationlimit];
      VTruncated = Take[V, All, truncationlimit];

      (* Reconstructing the compressed image *)
compressedImage = Image[UTruncated.DiagonalMatrix[sValuesTruncated].Transpose[VTruncated]];

      (* Adding the compressed image to the list *)
      AppendTo[compressedImages, compressedImage];
      (*calculating the compression ratio*)
      compressionRatio = N[(truncationlimit)/(Length[sValues])];
      AppendTo[comratio, compressionRatio];
      (*percentage compression*)
      percentagecompression = 100.0 (1 - (truncationlimit/Length[sValues]));
      AppendTo[comper,percentagecompression];
      (* Create a label for the compressed image *)
     label = StringJoin[
         "Number of Singular Values kept: " <> ToString[truncationlimit] <> "\n",
         "Compression Ratio: " <> ToString[compressionRatio]<> "\n",
        "Percentage of Compression: " <> ToString[percentagecompression]];

      (* Adding the label to the list *)
      AppendTo[labels, label],

      {truncationlimit, truncationlimitlist}
    ];

    (* Displaying the original and compressed images with labels *)
    Grid[Join[
  {{Magnify[ImageResize[imgGray, 300], 1.5], Magnify[ImageResize[img, 300], 1.5]}},
  Transpose[{Magnify[ImageResize[#, 300], 1.5] & /@ compressedImages, labels}]
  ], Dividers → All, Spacings→{2, 2}]
```

Out[•]=

	Number of Singular Values kept: 10 Compression Ratio: 0.00462963 Percentage of Compression: 99.537
	Number of Singular Values kept: 30 Compression Ratio: 0.0138889 Percentage of Compression: 98.6111
	Number of Singular Values kept: 50 Compression Ratio: 0.0231481 Percentage of Compression: 97.6852
	Number of Singular Values kept: 70 Compression Ratio: 0.0324074 Percentage of Compression: 96.7593
	Number of Singular Values kept: 90 Compression Ratio: 0.0416667 Percentage of Compression: 95.8333
	Number of Singular Values kept: 110 Compression Ratio: 0.0509259 Percentage of Compression: 94.9074

```
Number of Singular Values kept: 130
Compression Ratio: 0.0601852
Percentage of Compression: 93.9815
```

In[•]:=
```
ImageDimensions[img]
```

Out[•]=
```
{3840, 2160}
```

The percentage of compression will be less for low pixel density images. This is because the total number of pixels in the image determines the size of the singular value matrix, which in turn affects the amount of information that can be retained during the compression process.
In the case of a 4K image, the total number of pixels is very large, which means that the singular value matrix will also be large. This can make it difficult to achieve a high level of compression while still retaining enough information to maintain the quality of the image. In other words, it can be challenging to strike a balance between compression and image quality when working with high pixel density images.
On the other hand, if we work with low pixel density images, the total number of pixels will be lower, and the singular value matrix will be smaller. This can make it easier to achieve a higher level of compression while still retaining enough information to maintain the quality of the image. In other words, it can be easier to strike a balance between compression and image quality when working with low pixel density images.
Of course, the optimal level of compression will depend on a variety of factors, including the specific image being compressed, the desired level of image quality, and the available computing resources. It is always important to carefully consider these factors when performing image compression, regardless of the pixel density of the image.
SVD is also closely related to other matrix factorization techniques, such as principal component analysis (PCA). In fact, PCA can be seen as a special case of SVD, where the original matrix is centered and scaled to have zero mean and unit variance. PCA then uses the left singular matrix to identify the principal components of the original matrix.

Why SVD is considered as one of the most important tools in Data Analysis?

Singular Value Decomposition (SVD) is considered one of the powerful tools in data analysis for several reasons:

Dimensionality reduction: SVD can be used to reduce the dimensionality of a dataset by representing it in terms of a smaller number of dimensions that capture most of the variation in the original data. This is especially useful for large datasets where it can be difficult to analyze all the variables at once.

Noise reduction: SVD can help remove noise from a dataset by identifying the most important patterns and features that explain the variation in the data. By focusing on the most significant singular values and vectors, SVD can help filter out random noise that can obscure the underlying patterns in the data.

Compression: SVD can be used to compress data by representing it in terms of a smaller number of singular values and vectors. This can be useful for storing and transmitting large amounts of data more efficiently, without sacrificing too much information.

Data imputation: SVD can be used to fill in missing values in a dataset by predicting the values based on the remaining values and patterns in the data. This is useful for dealing with incomplete datasets, where some data points may be missing due to measurement errors, data entry errors, or other reasons.

Matrix approximation: SVD can be used to approximate a matrix by finding the closest possible approximation in terms of a lower rank matrix. This is useful for a wide range of applications, such as image processing, signal processing, and recommender systems.

Overall, SVD is a powerful tool in data analysis because it can help identify important patterns and features in the data, reduce noise, compress data, fill in missing values, and approximate matrices. These capabilities make it useful for a wide range of applications in fields such as data science, machine learning, image processing, and signal processing.

Why PCA is considered as a special case of SVD

Principal Component Analysis (PCA) is a technique used for dimensionality reduction, where it identifies the most important patterns and features in a dataset by finding the directions of maximum variance in the data. This is done by computing the eigenvectors and eigenvalues of the covariance matrix of the dataset. The eigenvectors represent the directions of maximum variance, and the corresponding eigenvalues represent the amount of variance explained by each direction.

Singular Value Decomposition (SVD) is a matrix factorization technique that decomposes a matrix into three matrices, U, Σ, and V^T. The diagonal matrix Σ contains the singular values, which are related to the eigenvalues of the covariance matrix of the data.

Now, the relationship between PCA and SVD can be seen as follows:

If we apply SVD to the centered data matrix, X, we get: X = UΣV^T
The columns of U represent the principal components of the data, which are the eigenvectors of the covariance matrix of X.
The singular values in Σ represent the square roots of the eigenvalues of the covariance matrix of X, multiplied by the square root of (n-1), where n is the number of observations in the data.
The columns of V represent the loadings of the principal components on the original variables.
Thus, we can see that the principal components computed by PCA are equivalent to the left singular vectors of SVD, and the amount of variance explained by each principal component is propor-

tional to the square of the corresponding singular value. Therefore, PCA can be seen as a special case of SVD, where we only need to compute the left singular vectors and singular values, and we do not need to compute the right singular vectors (i.e., the loadings). This makes PCA computationally more efficient than SVD in many cases.

Comparison between SVD and PCA

Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) are closely related, and SVD can be used to find PCA by truncating the less important basis vectors in the original SVD matrix.

In SVD, a matrix is decomposed into three matrices: U, Σ, and V^T, where U and V^T are orthogonal matrices, and Σ is a diagonal matrix of singular values. The singular values in Σ represent the amount of variation in the data that is captured by each corresponding singular vector in U and V^T.

PCA, on the other hand, seeks to identify the most important patterns and features in a dataset by finding a small number of linear combinations of the original variables that capture most of the variation in the data.

To find PCA using SVD, we can first compute the SVD of the data matrix. We can then truncate the less significant singular values and corresponding singular vectors in U and V^T. The remaining singular values and vectors can be used to reconstruct the data matrix with a reduced number of dimensions. The truncated matrix of singular vectors is equivalent to the matrix of principal components in PCA.

The advantage of using SVD to find PCA is that SVD provides a more general and robust framework for matrix factorization and dimensionality reduction. SVD can be applied to any matrix, not just to covariance or correlation matrices, and it is scale-invariant. SVD also provides a complete decomposition of the matrix, including information about the rank and null space of the matrix, which can be useful for many applications.