Vector Spaces

The following document contains screenshots of code cells and the output for this assignment.

Concept Exploration

To determine whether S is linearly independent, we need to solve the system of linear equations:

$$c_1v_1 + c_2v_2 + \ \dots + c_nv_n = 0,$$

where $c_1, c_2, \dots, c_n$ are coefficients that may or may not be zero. If the only solution to this system of equations is the trivial solution ($c_1 = c_2 = \ \dots = c_n = 0$, then S is linearly independent. Otherwise, S is linearly dependent.

To determine whether a vector ~w is in S, we need to solve the system of linear equations:

$$c_1v_1 + c_2v_2 + \ \dots + c_nv_n = \ vector(w),$$

where $c_1, c_2, \dots, c_n$ are coefficients. If there exists a solution to this system of equations, then vector (w) is in S. Otherwise, vector (w) is not in S.

The relationship between these two systems of equations is that determining whether a vector is in the span of a set of vectors (i.e., whether it can be expressed as a linear combination of those vectors) is equivalent to solving a system of linear equations. The coefficients of the solution to the second system of equations represent the coordinates of vector (w) in S, i.e., the coefficients $c_1, c_2, \dots, c_n$ are the coordinates of ~w with respect to the basis $\{v_1, v_2, \dots, v_n\}$.

Deep Dive

Q1:

The standard basis vectors, in this case, are:

$$e_1 = [1, 0, 0, 0, 0, 0, 0, 0]$$

$$e_2 = [0, 1, 0, 0, 0, 0, 0, 0]$$

$$e_3 = [0, 0, 1, 0, 0, 0, 0, 0]$$

$$e_4 = [0, 0, 0, 1, 0, 0, 0, 0]$$

$$e_5 = [0, 0, 0, 0, 1, 0, 0, 0]$$

$$e_6 = [0, 0, 0, 0, 0, 1, 0, 0]$$

$$e_7 = [0, 0, 0, 0, 0, 0, 1, 0]$$

$$e_8 = [0, 0, 0, 0, 0, 0, 0, 1]$$

Each of these basis vectors corresponds to a single pixel, with a value of 1 in the position corresponding to that pixel and a value of 0 elsewhere.

A linear combination of the standard basis vectors represents a weighted sum of the corresponding pixels in the image. For example, the vector [31, 159, 0, 0, 0, 0, 0, 0] is a linear combination of the first two standard basis vectors, where the first pixel is weighted by 31 and the second pixel is weighted by 159. This vector represents an image that consists only of the first two pixels, with all other pixels set to 0.

The coordinates of a vector represent the weights of the corresponding pixels in terms of the standard basis vectors. For example, the vector v = [31, 159, 9, 162, 233, 54, 217, 3] can be written as a linear combination of the standard basis vectors as follows:

$$v = 31e_1 + 159e_2 + 9e_3 + 162e_4 + 233e_5 + 54e_6 + 217e_7 + 3e_8$$

A vector's linear combinations are expressed through the coefficients of its coordinates. This can be visualized as the weighting given to each light intensity in an image, which, in turn, determines the precise value of the pixel's light intensity relative to the standard basis vectors.
The coefficients of this linear combination, i.e., 31, 159, 9, 162, 233, 54, 217, and 3, are the coordinates of the vector v with respect to the standard basis vectors.

Q2:

(a)

In the realm of linear algebra, an orthogonal basis comprises a collection of vectors that wholly span the vector space they are situated in. Any two given vectors belonging to this set lie perpendicular relative to each other. This indicates that no member within the said group can be represented as some combination derived from any others, which is most evident when one visually inspects these various directional quantities.

To show that the vectors $v_1, v_2, \dots, v_8$ form an orthogonal basis of R^8, we need to show two things: first, that they are linearly independent, and second, that they span R^8.

To show linear independence, we will show that the only linear combination of these vectors that yields the zero vector is the trivial combination. That is, we want to show that if

$a_1v_1 + a_2v_2 + \dots + a_8v_8 = 0$, then $a_1 = a_2 = \dots = a_8 = 0$.

Given that $a_1v_1 + a_2v_2 + \dots + a_8v_8 = 0$. This suggests that every element within each vector on the left side is equal to zero. Consequently, this leads us towards having an arrangement of equations as follows:

$$a_1 + a_2 + a_3 + a_4 = 0$$

$$a_1 + a_2 + a_3 + a_4 - a_5 - a_6 - a_7 - a_8 = 0$$

$$a_1 + a_2 - a_3 - a_4 = 0$$

$$-a_4 - a_8 = 0$$

$$-a_4 - a_8 = 0$$

$$a_1 - a_2 = 0$$

$$a_3 - a_4 = 0$$

$$a_5 - a_6 = 0$$

$$a_7 - a_8 = 0$$

Or using Sage we can show it as:

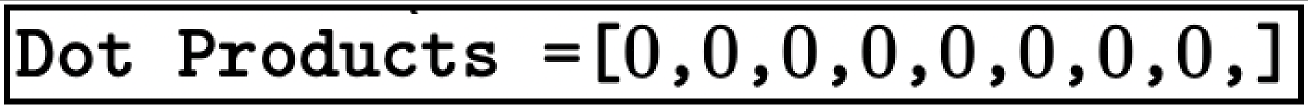
```
#Defining Vectors
v1 = vector(QQ,[1]*8)
v2 = vector(QQ, [1]*4+[-1]*4)
v3 = vector(QQ, [1]*2+[-1]*2+[0]*4)
v4 = vector(QQ,[0]*4+[1]*2+[-1]*2)
v5 = vector(QQ,[1]+[-1]+[0]*6)
v6 = vector(QQ,[0]*2+[1]+[-1]+[0]*4)
v7 = vector(QQ,[0]*4+[1]+[-1]+[0]*2)
v8 = vector(QQ,[0]*6+[1]+[-1])
v = vector(QQ,[31,159,9,162,233,54,217,3])


#Checking Span
S = (matrix(QQ, [v1,v2,v3,v4,v5,v6,v7,v8]).transpose().augment(v,
subdivide = True))

pretty_print("S =", S)

pretty_print("S-RREF =", S.rref())
```

$$\mathtt{S}\ = \left(\begin{array}{cccccccc|c}
1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 31 \\
1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 159 \\
1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 9 \\
1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 & 162 \\
1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 & 233 \\
1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 & 54 \\
1 & -1 & 0 & -1 & 0 & 0 & 0 & 1 & 217 \\
1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 & 3
\end{array}\right)$$

$$\mathtt{S\text{-}RREF}\ = \left(\begin{array}{cccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{217}{2} \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{73}{4} \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \frac{19}{4} \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \frac{67}{4} \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -64 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -\frac{153}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \frac{179}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 107
\end{array}\right)$$

The reason why the dot product of any two orthogonal vectors is zero is that they are at right angles to each other, or perpendicular. Therefore, to verify if the vectors in set S are orthogonal, we can compute their dot products. By calculating the dot product for every possible pair of vectors in S, we found that it was always equal to zero. Below is a list of the dot products for each unique pair of vectors in S.

```
#computing dot product of all pairs in S
vectors = [v1,v2,v3,v4,v5,v6,v7,v8]
dot_products = []
for i in range(len(vectors)):
    for j in range(len(vectors)):
        if i != j:
            x = vectors[i]
            y = vectors[j]
            dot_products.append(x.dot_product(y))
pretty_print("Dot Products =","[",dot_products[0],",",
dot_products[1],",", dot_products[2],",", dot_products[3],",",
             dot_products[4],",", dot_products[5],",",
dot_products[6],",", dot_products[7],",","]")
```



```
Dot Products =[0,0,0,0,0,0,0,0,]
```

Given that all orthogonal vectors are linearly independent, hence the vectors in S form orthogonal basis.

(b)

The grayscale pixels in an image can be represented by linear combinations of the basis given in S. A grayscale image can be represented by a vector v where each row corresponds to the intensity value of a pixel. By manipulating the linear combinations of the vectors in v, the light intensity of each pixel in the image can be adjusted. This process results in different vectors representing grayscale images with unique light-intensity patterns.

The code below and Fig. 1 show the representation of linear combinations of basis vectors in S.

```
#plotting matrix plot
matrix_plot(matrix([v1, 100*v1 + 50*v2, 128*v1 - 64*v3 + 32*v5 - 16*v7]),
cmap ='gray')
```

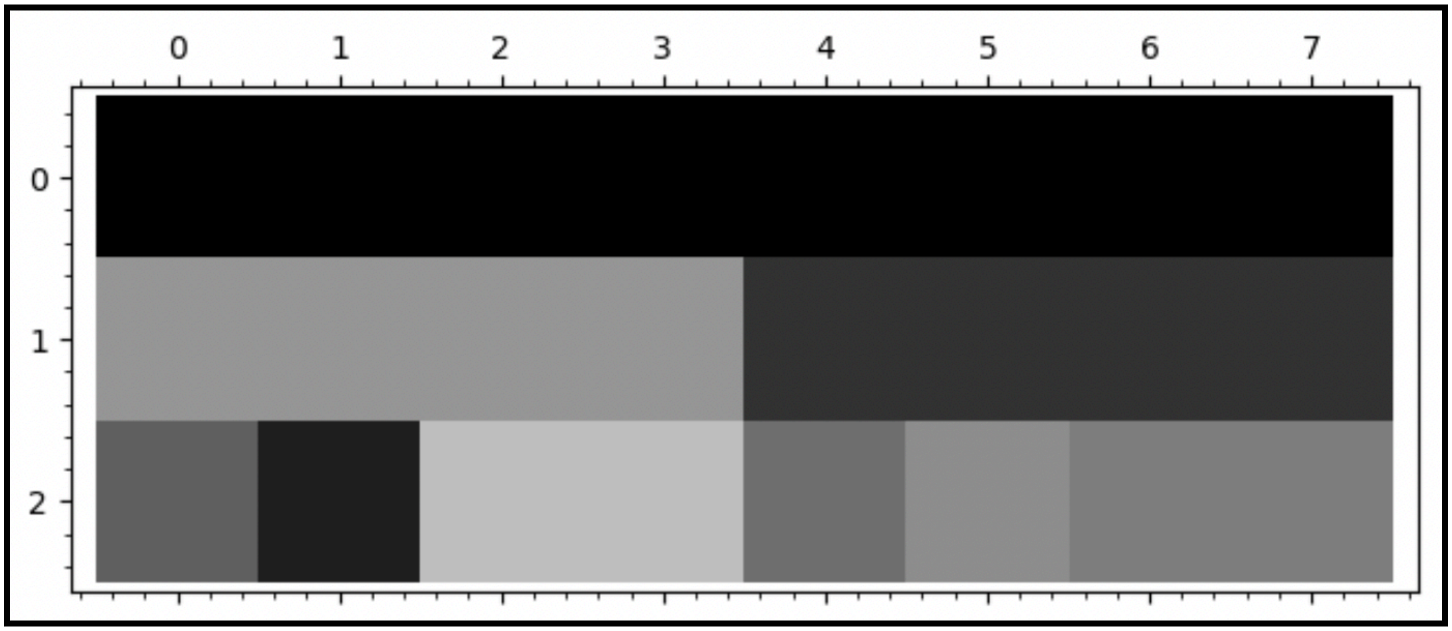


Fig.1 shows grayscale images of linear combinations of basis from vector space S.

Here are the grayscale images corresponding to v1, 100v1 + 50v2, and 128v1 − 64v3 + 32v5 − 16v7:

The evidence presented in Figure 1 demonstrates that a black-and-white picture has the ability to be expressed through an assortment of basis vectors, namely v1, v2..., and v8. Each grayscale image can actually be separated into eight discreet parts with individual orthogonal properties which correspond directly to one particular vector. The components corresponding to v1 and v2 are the global intensity and the horizontal gradient, respectively, while the other components represent various forms of edges and textures in the image. By adjusting the coefficients of the linear combination, one can emphasize or suppress certain features of the image, or create new images by combining the features of different images.

As a whole, the set of eight vectors (v1 through v8) creates an independent foundation for grayscale image space. With this in mind, all grayscale images can be expressed as combinations made linearly from these designated vectors. This approach enables intricate and adaptable representations that are useful in diverse applications like feature extraction, compression or picture manipulation.

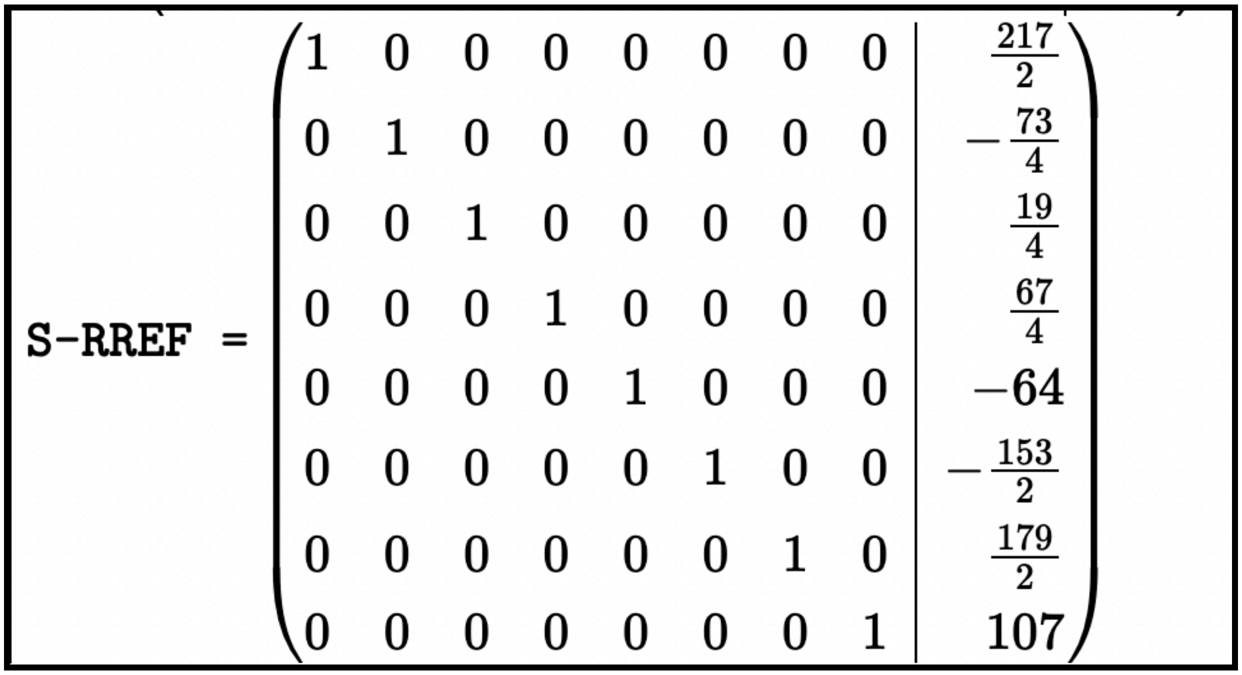
Q3:

(a)

We can find the coordinates by transposing matrix vectors from S to form columns in a matrix and augmenting the matrix with vector v = [31, 159, 0, 0, 0, 0, 0, 0]. After this, we can take the RREF and RHS will show the coordinates as shown below.

```
#Defining Vectors
v1 = vector(QQ,[1]*8)
v2 = vector(QQ, [1]*4+[-1]*4)
v3 = vector(QQ, [1]*2+[-1]*2+[0]*4)
v4 = vector(QQ,[0]*4+[1]*2+[-1]*2)
v5 = vector(QQ,[1]+[-1]+[0]*6)
v6 = vector(QQ,[0]*2+[1]+[-1]+[0]*4)
```

```
v7 = vector(QQ,[0]*4+[1]+[-1]+[0]*2)
v8 = vector(QQ,[0]*6+[1]+[-1])
v = vector(QQ,[31,159,9,162,233,54,217,3])


#Checking Span
S = (matrix(QQ, [v1,v2,v3,v4,v5,v6,v7,v8]).transpose().augment(v,
subdivide = True))

pretty_print("S-RREF =", S.rref())
```



$$\text{S-RREF} = \left(\begin{array}{cccccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{217}{2} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{73}{4} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \frac{19}{4} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \frac{67}{4} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -64 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -\frac{153}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \frac{179}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 107 \end{array}\right)$$

Hence, the RHS shows the coordinates of v in terms of the new basis S.

(b)

Let's assign some threshold values.

$$\varepsilon_1 = 10$$

$$\varepsilon_2 = 20$$

$$\varepsilon_3 = 30$$

The vectors c_1, c_2 *and* c_3 correspond to three distinct compressed variants of the grayscale image. In these compressed versions, any pixel with a light intensity below a certain threshold has been eliminated and assigned a value of 0, resulting in a black color with zero brightness level.

```
# Define the vector v and the thresholds
v = vector(QQ, [217/2,-73/4,19/4,67/4,-64,-153/2,179/2,107])
thresholds = [10, 20, 30]

# Create a list to hold the compressed vectors
compressed_vectors = []

# Loop over the thresholds to compute the compressed vectors
for threshold in thresholds:
    compressed_vector = zero_vector(QQ, len(v)) # Initialize the
compressed vector with all zeros

    # Loop over the entries of the original vector and compress the ones
with magnitude greater than or equal to the threshold
    for i in range(len(v)):
        if abs(v[i]) >= threshold:
            compressed_vector[i] = v[i]

    compressed_vectors.append(compressed_vector)

C1 = matrix(QQ, [compressed_vectors[0]]).transpose()
C2 = matrix(QQ, [compressed_vectors[1]]).transpose()
C3 = matrix(QQ, [compressed_vectors[2]]).transpose()

pretty_print("C1 =",C1,"C2 =",C2,"C3 =",C3)
```

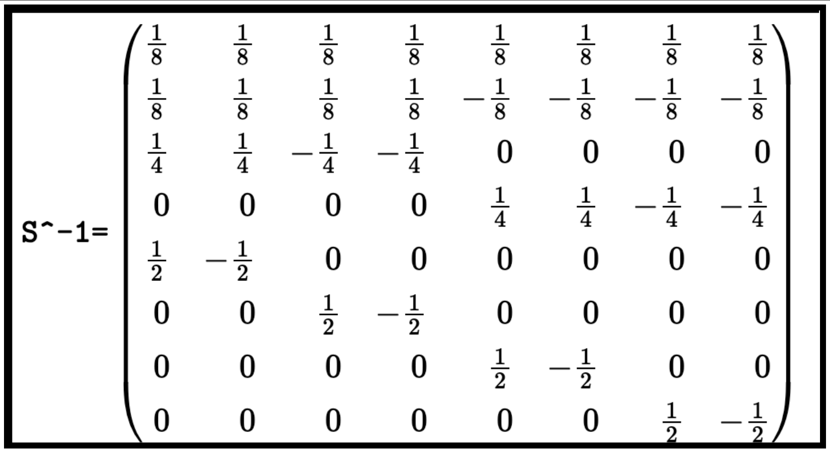
$$\mathtt{C1} = \begin{pmatrix} \frac{217}{2} \\ -\frac{73}{4} \\ 0 \\ \frac{67}{4} \\ -64 \\ -\frac{153}{2} \\ \frac{179}{2} \\ 107 \end{pmatrix} \quad \mathtt{C2} = \begin{pmatrix} \frac{217}{2} \\ 0 \\ 0 \\ 0 \\ -64 \\ -\frac{153}{2} \\ \frac{179}{2} \\ 107 \end{pmatrix} \quad \mathtt{C3} = \begin{pmatrix} \frac{217}{2} \\ 0 \\ 0 \\ 0 \\ -64 \\ -\frac{153}{2} \\ \frac{179}{2} \\ 107 \end{pmatrix}$$

Where c_1, c_2 *and* c_3 are compressed by $\varepsilon_1, \varepsilon_2$ *and* ε_3 respectively.

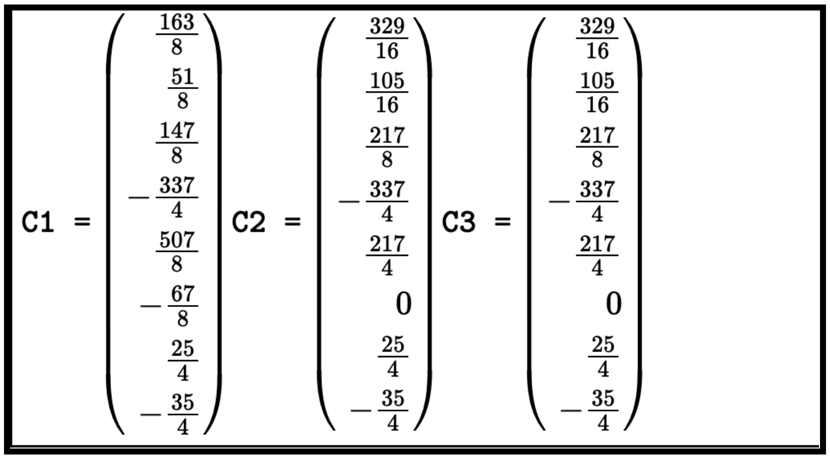
To find the standard basis we need to have a change of basis matrix. We can use S as our change of basis matrix and multiply the inverse of S with each compressed vector c_1, c_2 *and* c_3 to update c_1, c_2 *and* c_3 as standard basis as shown below.

```
#Defining Vectors
v1 = vector(QQ,[1]*8)
v2 = vector(QQ, [1]*4+[-1]*4)
v3 = vector(QQ, [1]*2+[-1]*2+[0]*4)
v4 = vector(QQ,[0]*4+[1]*2+[-1]*2)
v5 = vector(QQ,[1]+[-1]+[0]*6)
v6 = vector(QQ,[0]*2+[1]+[-1]+[0]*4)
v7 = vector(QQ,[0]*4+[1]+[-1]+[0]*2)
v8 = vector(QQ,[0]*6+[1]+[-1])
v = vector(QQ,[31,159,9,162,233,54,217,3])

#Checking Span
S = (matrix(QQ, [v1,v2,v3,v4,v5,v6,v7,v8]).transpose())
```

```
S_inverse = S.inverse()
pretty_print(S_inverse)

# Define the three compressed vectors C1, C2, and C3
C1 = matrix(QQ, [compressed_vectors[0]]).transpose()
C2 = matrix(QQ, [compressed_vectors[1]]).transpose()
C3 = matrix(QQ, [compressed_vectors[2]]).transpose()

#defining standard basis
Standard_C1 = S_inverse * C1
Standard_C2 = S_inverse * C2
Standard_C3 = S_inverse * C3

pretty_print("C1 =",Standard_C1,"C2 =",Standard_C2,"C3 =",Standard_C3)
```



$$\texttt{S^-1=}\begin{pmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} \end{pmatrix}$$



$$\texttt{C1 =}\begin{pmatrix} \frac{163}{8} \\ \frac{51}{8} \\ \frac{147}{8} \\ -\frac{337}{4} \\ \frac{507}{8} \\ -\frac{67}{8} \\ \frac{25}{4} \\ -\frac{35}{4} \end{pmatrix} \texttt{C2 =}\begin{pmatrix} \frac{329}{16} \\ \frac{105}{16} \\ \frac{217}{8} \\ -\frac{337}{4} \\ \frac{217}{4} \\ 0 \\ \frac{25}{4} \\ -\frac{35}{4} \end{pmatrix} \texttt{C3 =}\begin{pmatrix} \frac{329}{16} \\ \frac{105}{16} \\ \frac{217}{8} \\ -\frac{337}{4} \\ \frac{217}{4} \\ 0 \\ \frac{25}{4} \\ -\frac{35}{4} \end{pmatrix}$$

Now we can plot the results either as a 3-dimensional figure or in matrix plot to compare the results.

```
# Plot the vectors
p = arrow3d((0,0,0), Standard_C1.list()[:3], color='red', size=10)
q = arrow3d((0,0,0), Standard_C2.list()[:3], color='green', size=10)
r = arrow3d((0,0,0), Standard_C3.list()[:3], color='blue', size=10)
show(p+q+r, aspect_ratio=(1,1,1))

matrix_plot(matrix([Standard_C1, Standard_C2, Standard_C3]), cmap
='gray')
```

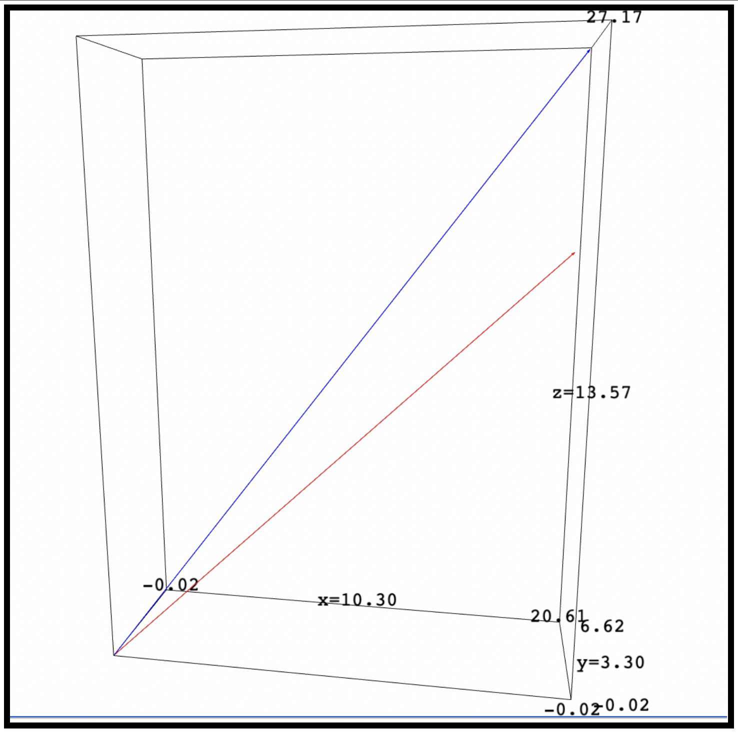


Fig.2 shows the standard vectors for c1, c2 and c3, forming a triangular plane between red and blue lines.

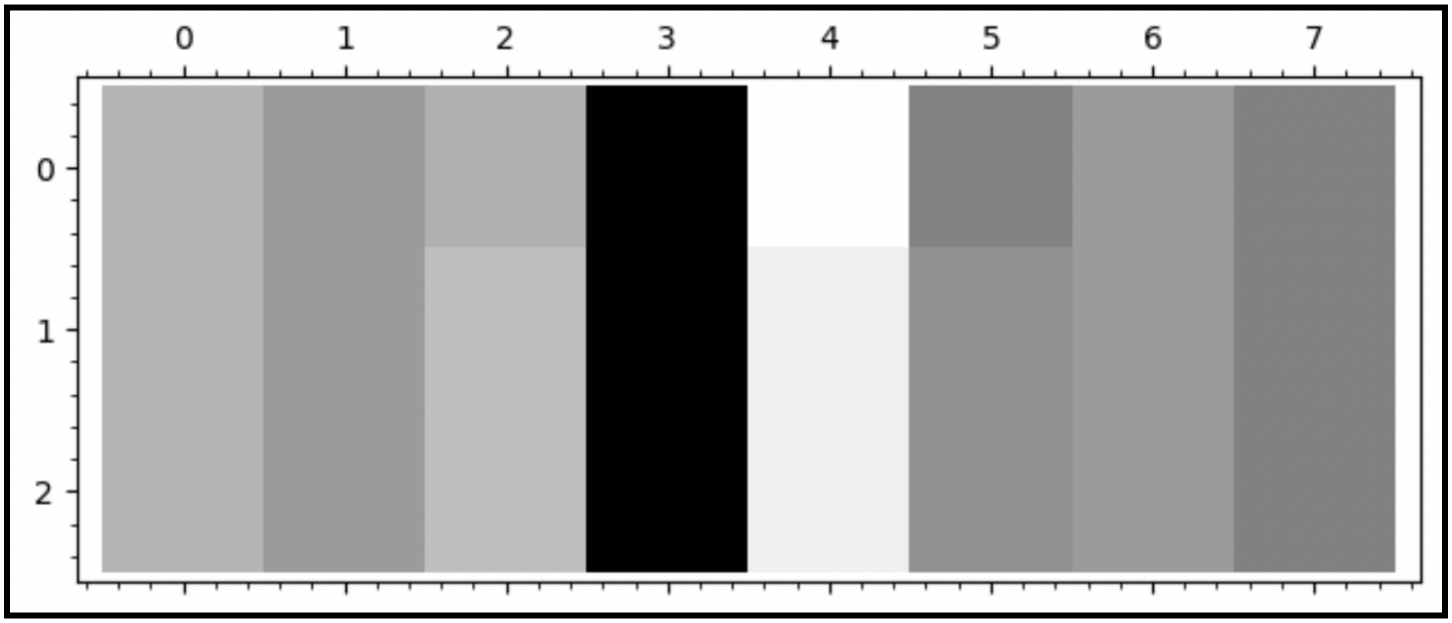


Fig.3 shows grayscale images of standard basis from vector space S.

The 3D arrow plot shows three vectors in red, green, and blue. These vectors represent the compressed versions of the original vector v with respect to different thresholds. The x, y, and z components of the vectors are given by the first three entries of the vectors. The three vectors are plotted from the origin (0, 0, 0) to their respective endpoints. The red vector represents the compressed version of v using a threshold of 10, the green vector represents the compressed version of v using a threshold of 20, and the blue vector represents the compressed version of v using a threshold of 30. The plot can be used to visualize how the compression changes as the threshold increases.

The matrix plot shows the three compressed vectors as grayscale images. The horizontal axis represents the entries of the compressed vectors, and the vertical axis represents the three different compressed vectors. The darker the color of a pixel, the larger the magnitude of the corresponding entry of the compressed vector. The plot can be used to visualize the structure of the compressed vectors and how they differ from each other.

The matrix plot helps us to visualize the amount of information preserved in the compressed vectors for different levels of compression. As the compression threshold increases, more entries in the original vector v are set to zero, resulting in a more compressed vector. This leads to a loss of information, as some of the original entries are discarded. In the matrix plot, we can observe that the compressed vector C1, which is obtained by setting the compression threshold to 10, has the darkest pixels, indicating that it preserves the most information among the three compressed vectors. The compressed vector C3, which is obtained by setting the compression threshold to 30, has the lightest pixels, indicating that it preserves the least information.

(c)

To experiment with choices of the threshold value, a list of compressed vectors in the basis representing compression according to threshold values ranging from 10 to 30 and then 100 to 110 was generated. These vectors were then converted into standard basis and plotted against the original vector

```
#Defining Vectors
v1 = vector(QQ,[1]*8)
v2 = vector(QQ, [1]*4+[-1]*4)
v3 = vector(QQ, [1]*2+[-1]*2+[0]*4)
v4 = vector(QQ,[0]*4+[1]*2+[-1]*2)
v5 = vector(QQ,[1]+[-1]+[0]*6)
v6 = vector(QQ,[0]*2+[1]+[-1]+[0]*4)
v7 = vector(QQ,[0]*4+[1]+[-1]+[0]*2)
v8 = vector(QQ,[0]*6+[1]+[-1])
v = vector(QQ,[31,159,9,162,233,54,217,3])

#Checking Span
S = (matrix(QQ, [v1,v2,v3,v4,v5,v6,v7,v8]).transpose())
S_inverse = S.inverse()

# Define the vector v and the thresholds
```

```
v = vector(QQ, [217/2,-73/4,19/4,67/4,-64,-153/2,179/2,107])
thresholds = [10, 20, 30, 40, 50,100, 108, 110]

# Create a list to hold the compressed vectors
compressed_vectors = []

# Loop over the thresholds to compute the compressed vectors
for threshold in thresholds:
    compressed_vector = zero_vector(QQ, len(v)) # Initialize the compressed vector with
all zeros

    # Loop over the entries of the original vector and compress the ones with magnitude
greater than or equal to the threshold
    for i in range(len(v)):
        if abs(v[i]) >= threshold:
            compressed_vector[i] = v[i]

    compressed_vectors.append(compressed_vector)

# Define the three compressed vectors C1, C2, and C3
C1 = matrix(QQ, [compressed_vectors[0]]).transpose()
C2 = matrix(QQ, [compressed_vectors[1]]).transpose()
C3 = matrix(QQ, [compressed_vectors[2]]).transpose()
C4 = matrix(QQ, [compressed_vectors[3]]).transpose()
C5 = matrix(QQ, [compressed_vectors[4]]).transpose()
C6 = matrix(QQ, [compressed_vectors[5]]).transpose()
C7 = matrix(QQ, [compressed_vectors[6]]).transpose()
C8 = matrix(QQ, [compressed_vectors[7]]).transpose()



#defining standard basis
Standard_v10 = S_inverse * C1
Standard_v20 = S_inverse * C2
Standard_v30 = S_inverse * C3
Standard_v40 = S_inverse * C4
Standard_v50 = S_inverse * C5
Standard_v100 = S_inverse * C6
Standard_v108 = S_inverse * C7
Standard_v110 = S_inverse * C8

# Plot the vectors
```

```
v10 = vector(QQ,[163/8, 51/8,147/8,-337/4,507/8,-67/8,25/4,-35/4])
v20 = vector(QQ,[329/16,105/16,217/8,-337/4,217/4,0,25/4,-35/4])
v30 = vector(QQ,[329/16,105/16,217/8,-337/4,217/4,0,25/4,-35/4])
v40 = vector(QQ,[329/16,105/16,217/8,-337/4,217/4,0,25/4, -35/4])
v50 = vector(QQ,[329/16,105/16,217/8,-337/4,217/4,0,25/4, -35/4])
v100 = vector(QQ,[431/16,3/16,217/8,-107/4,217/4,0,0,-107/2])
v108 = vector(QQ,[217/16,217/16,217/8,0,217/4,0,0,0])
v110 = vector(QQ,[0,0,0,0,0,0,0,0])


P1 = matrix_plot(matrix([v,v10,v20,v30]), cmap ='gray', colorbar = True,
colorbar_orientation = "horizontal")
pretty_print("P1 ="),show(P1)
P2 = matrix_plot(matrix([v, v100, v108, v110]), cmap ='gray',colorbar =
True,colorbar_orientation = "horizontal")
pretty_print("P2 ="),show(P2)
```

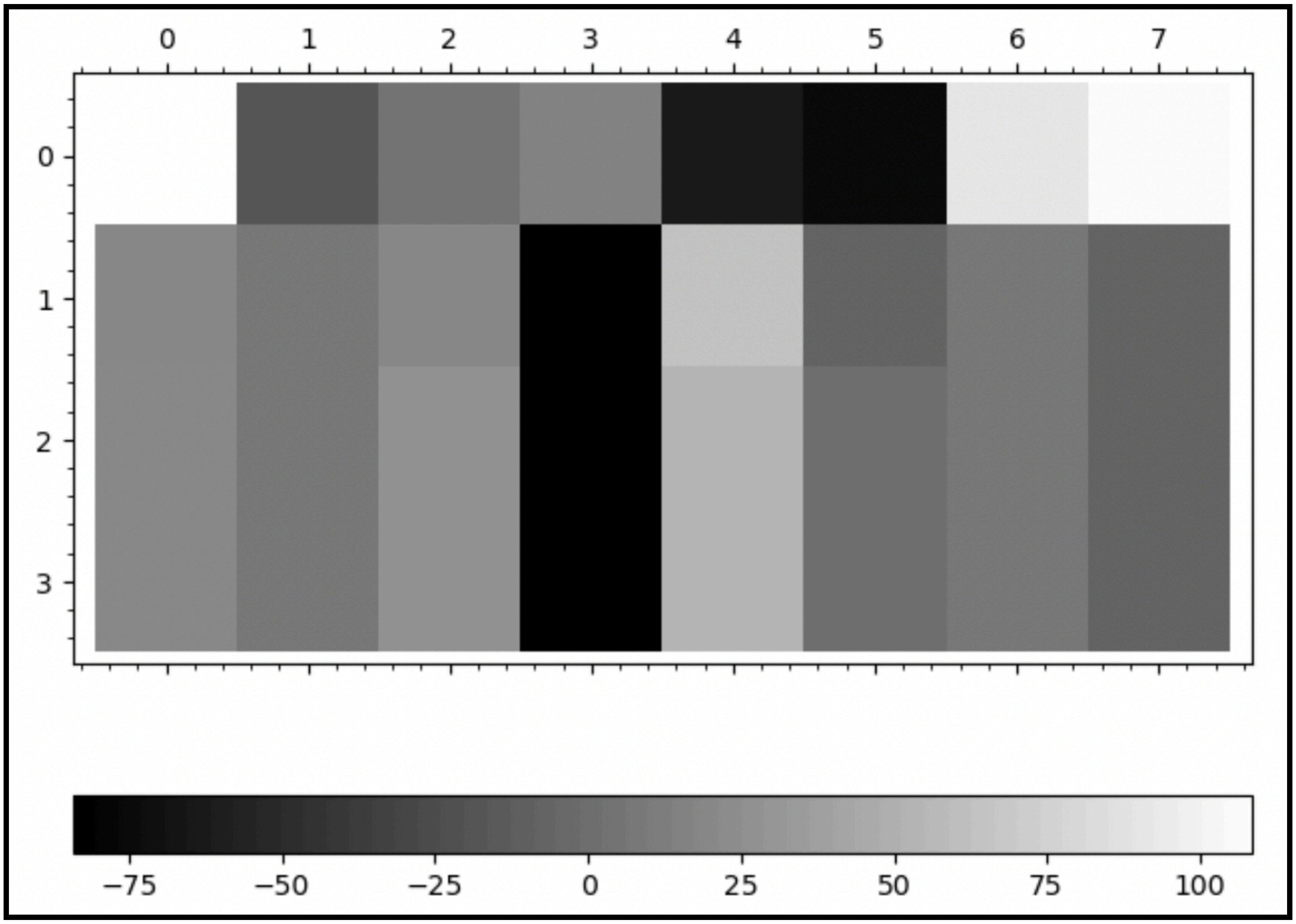


Fig.4 shows a grayscale image of vectors when the threshold was between 10 to 30.

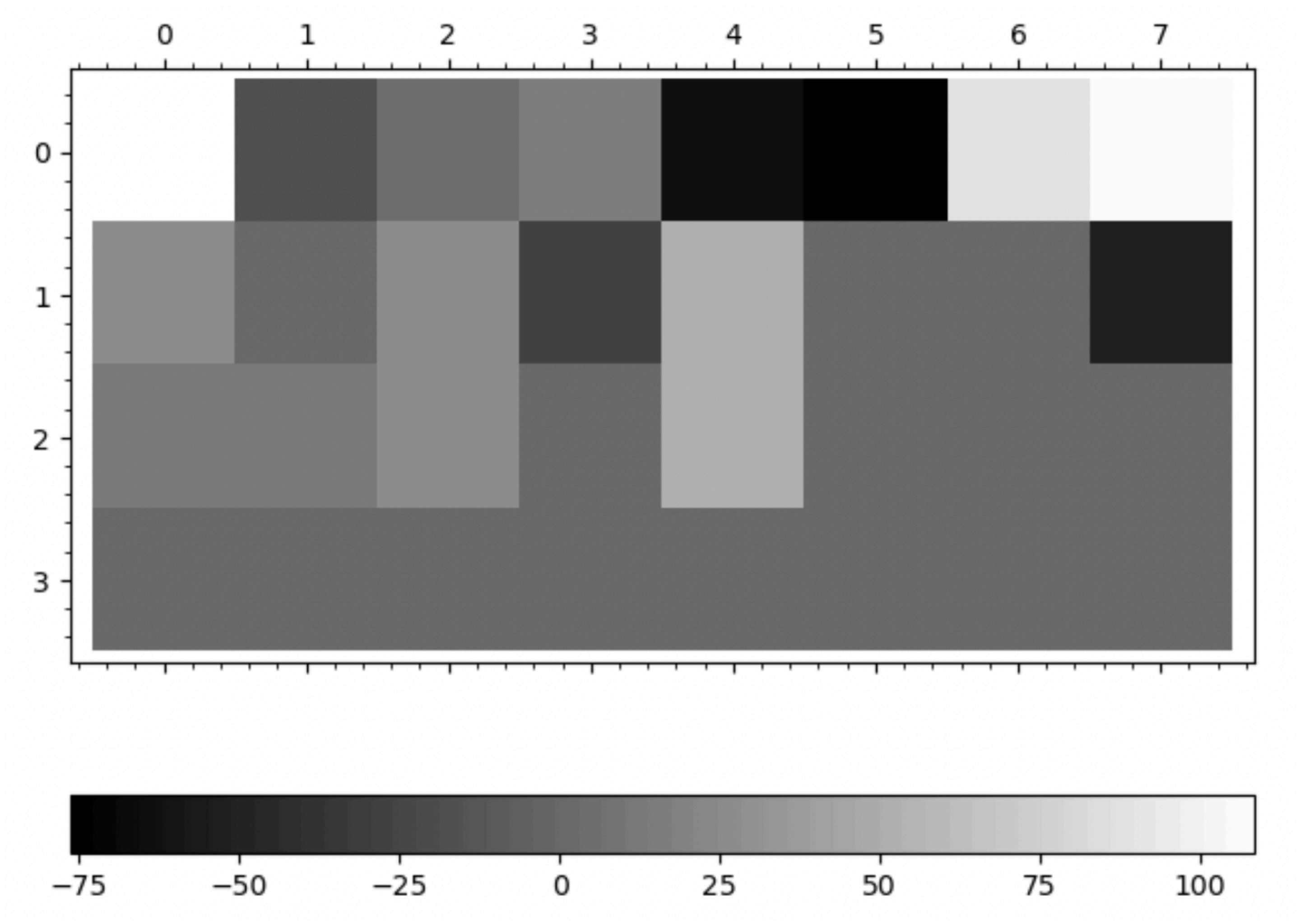


Fig. 5 shows a grayscale image of vectors when the threshold was between 100, 108 and 110.

Interpretation:

When we printed the results, we noticed that after the threshold of 108, we lose all the information as shown by the last row in Fig. 5 at the 110 thresholds. We experimented with multiple values but saw a noticeable difference when we were increasing the threshold from 10 to 30 compared to 100 to 110. As after 108, the image is fully compressed. For a small noticeable difference, we expect to see a noticeable change between thresholds of 10 and 20 as from Fig. 4, we can see that there is a noticeable difference between rows 2 and 3. We can further zoom in to see the difference as shown below:

```
Standard_v14 = S_inverse * C1
Standard_v16 = S_inverse * C2
Standard_v18 = S_inverse * C3

# Plot the vectors
v14 = vector(QQ,[163/8, 51/8, 147/8, -337/4, 507/8, -67/8, 25/4, -35/4])
v16 = vector(QQ,[163/8, 51/8, 147/8, -337/4, 507/8, -67/8, 25/4, -35/4])
v18 = vector(QQ,[585/3,137/32, 361/16,-337/4,507/8,0,25/4,-35/4])


P1 = matrix_plot(matrix([v,v14,v16,v18]), cmap ='gray', colorbar = True,
colorbar_orientation = "horizontal")
pretty_print("P1 ="),show(P1)
```

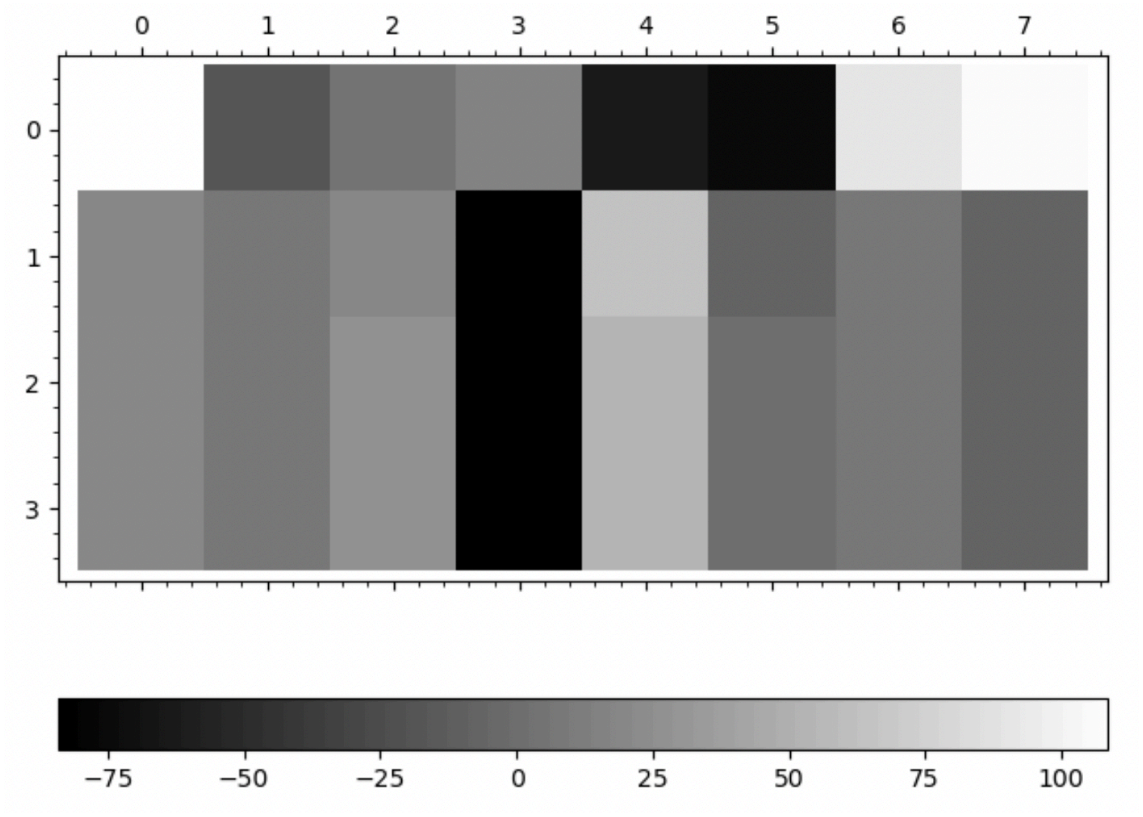


Fig. 6 shows a grayscale image of vectors when the threshold was between 14, 16 and 18.

Fig. 6 confirms that the noticeable difference becomes apparent after the threshold of 16.