Deep Dive -Vector Spaces

March 10, 2023

0.0.1 Concept Exploration

Given the set of vectors $S = \{v_1, v_2, \dots, v_n\}$

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

$$M\vec{x} = \vec{0}$$

where M is the matrix whose columns are the vectors in S and $\vec{0}$ is the zero vector.

The matrix M is given by:

$$M = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ v_1 & v_2 & \dots & v_n \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

To find the coordinates of the vector $\vec{w}$ in S, we need to solve the following system of equations:

$$M\vec{x} = \vec{w}$$

where M is the matrix whose columns are the vectors in S and $\vec{w}$ is the vector we want to find the coordinates of. The coordinates of $\vec{w}$ in S are given by the solution of the system of equations which is $\vec{x}$.

0.0.2 1

In terms of this image, the eight standard basis vectors represent the brightness levels of each pixel if the other pixels are all black. For example, the first standard basis vector is:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

which represents the intensity level of the first pixel if the other pixels are all black.

A linear combination of the eight standard basis vectors should produce the intensity level of each pixel in the 1×8 image. For example, the linear combination:

$$1\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}+1\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}+1\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}+1\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}+1\begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}+1\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix}+1\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}+1\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}=\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\\1\end{bmatrix}$$

should produce an image that is all black because each pixel in the resulting image has an intensity level of 1. See the image from the sage codecell below for the result of the linear combination above.

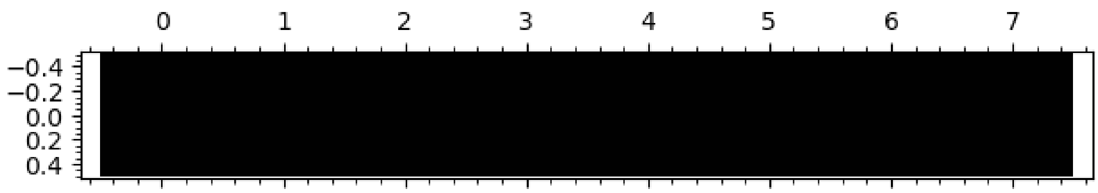
Note: The vectors above are represented as column vectors for clarity. In the context of the image, they are actually row vectors.

The vector formed by the linear combination represents the coordinates of the vector in the standard basis. As such, the coordinates of the vector is the brightness level of each pixel in the image.

[1]:
```
# standard basis vectors for R^8
e1 = vector([1,0,0,0,0,0,0,0])
e2 = vector([0,1,0,0,0,0,0,0])
e3 = vector([0,0,1,0,0,0,0,0])
e4 = vector([0,0,0,1,0,0,0,0])
e5 = vector([0,0,0,0,1,0,0,0])
e6 = vector([0,0,0,0,0,1,0,0])
e7 = vector([0,0,0,0,0,0,1,0])
e8 = vector([0,0,0,0,0,0,0,1])

# all pixels with a brightness of 1
combination = e1 + e2 + e3 + e4 + e5 + e6 + e7 + e8
print(combination)

def show_image(combination, title=None):
    '''
    Displays an grayscale image given a matrix of pixel values.

    Parameters
    ----------
    combination : matrix
        A a matrix of pixel intensity values.

    Returns
    -------
    None.
    '''
```

```
    return show(matrix_plot(matrix(combination), cmap="gray", vmax=255,␣
↪vmin=0), title=title)

show_image(combination)
```

```
(1, 1, 1, 1, 1, 1, 1, 1)
```



0.0.3 2

(a)

Given the eigth vectors

$$v_1 = \begin{bmatrix}1\\1\\1\\1\\1\\1\\1\\1\end{bmatrix}, v_2 = \begin{bmatrix}1\\1\\1\\1\\-1\\-1\\-1\\-1\end{bmatrix}, v_3 = \begin{bmatrix}1\\1\\-1\\-1\\0\\0\\0\\0\end{bmatrix}, v_4 = \begin{bmatrix}0\\0\\0\\0\\1\\1\\-1\\-1\end{bmatrix}, v_5 = \begin{bmatrix}1\\-1\\0\\0\\0\\0\\0\\0\end{bmatrix}, v_6 = \begin{bmatrix}0\\0\\1\\-1\\0\\0\\0\\0\end{bmatrix}, v_7 = \begin{bmatrix}0\\0\\0\\0\\1\\-1\\0\\0\end{bmatrix}, v_8 = \begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\-1\end{bmatrix}$$

To show that these vectors form an orthogonal basis of $\mathbb{R}^8$, we can add vectors as the column vectors of a matrix A and then check if $A^TA = X$ where X is a diagonal matrix.

$$A = \begin{bmatrix}\vdots & \vdots & \vdots & \vdots\\ v_1 & v_2 & \dots & v_n\\ \vdots & \vdots & \vdots & \vdots\end{bmatrix}$$

$$A^TA = \begin{bmatrix}v_1^T\\ v_2^T\\ \vdots\\ v_n^T\end{bmatrix}\begin{bmatrix}\vdots & \vdots & \vdots & \vdots\\ v_1 & v_2 & \dots & v_n\\ \vdots & \vdots & \vdots & \vdots\end{bmatrix} = \begin{bmatrix}v_1^Tv_1 & v_1^Tv_2 & \dots & v_1^Tv_n\\ v_2^Tv_1 & v_2^Tv_2 & \dots & v_2^Tv_n\\ \vdots & \vdots & \ddots & \vdots\\ v_n^Tv_1 & v_n^Tv_2 & \dots & v_n^Tv_n\end{bmatrix}$$

This works because the columns of A are the same as the rows of A^T. As such we can check if the dot product of all the vectors are either 0 or not zero. The diagonals represent the dot product of a vector with itself. As such, the diagonals should all be non-zero values and the off-diagonals should all be 0.

The result of $A^T A$ is (see the sage codecell below):

$$\begin{bmatrix} 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

From this we can see that the diagonals are all non-zero values and the off-diagonals are all 0. As such, the vectors form an orthogonal basis of $\mathbb{R}^8$.

Additionally, we can check the RREF of A to see if the vectors form a linearly independent set. The RREF of A is (see the sage codecell below):

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

As we can see, the RREF is the identity matrix. As such, the vectors form a linearly independent set. Since all vectors are linearly independent, they form a basis of $\mathbb{R}^8$.

[21]:
```
# new basis we are working with
v1 = vector([1,1,1,1,1,1,1,1])
v2 = vector([1,1,1,1,-1,-1,-1,-1])
v3 = vector([1,1,-1,-1,0,0,0,0])
v4 = vector([0,0,0,0,1,1,-1,-1])
v5 = vector([1,-1,0,0,0,0,0,0])
v6 = vector([0,0,1,-1,0,0,0,0])
v7 = vector([0,0,0,0,1,-1,0,0])
v8 = vector([0,0,0,0,0,0,1,-1])

# adding the vectors as the column vectors of a matrix A
A = matrix([v1,v2,v3,v4,v5,v6,v7,v8]).transpose()

# matrix transpose
A_transpose = A.transpose()

print(f"The result of the matrix multiplication of A_transpose and A is:
 ↪\n{A_transpose*A}")
```

```
The result of the matrix multiplication of A_transpose and A is:
[8 0 0 0 0 0 0 0]
[0 8 0 0 0 0 0 0]
[0 0 4 0 0 0 0 0]
[0 0 0 4 0 0 0 0]
[0 0 0 0 2 0 0 0]
[0 0 0 0 0 2 0 0]
[0 0 0 0 0 0 2 0]
[0 0 0 0 0 0 0 2]
```

[3]:
```python
print(f"The rref of A is:\n{A.rref()}")
```

```
The rref of A is:
[1 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0]
[0 0 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0]
[0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 1]
```

(b)

v_1 is a grayscale image with all pixels at intensity of 1.

Computing $100v_1 + 50v_2$,

$$100v_1 + 50v_2 = \begin{bmatrix}100\\100\\100\\100\\100\\100\\100\\100\end{bmatrix} + \begin{bmatrix}50\\50\\50\\50\\-50\\-50\\-50\\-50\end{bmatrix} = \begin{bmatrix}150\\150\\150\\150\\50\\50\\50\\50\end{bmatrix}$$

This is a grayscale image with the first four pixels at intensity of 150 and the last four pixels at intensity of 50.

Computing $128v_1 - 64v_3 + 32v_5 - 16v_7$, we get

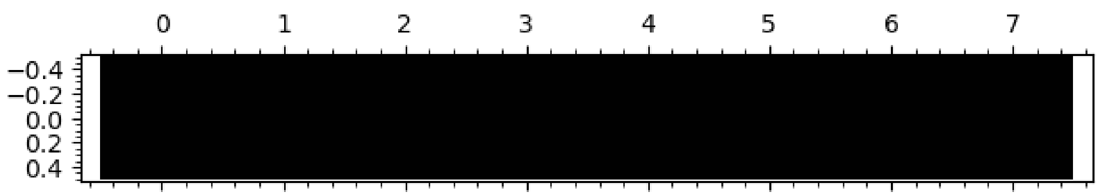
$$128v_1 - 64v_3 + 32v_5 - 16v_7 = \begin{bmatrix}128\\128\\128\\128\\128\\128\\128\\128\end{bmatrix} - \begin{bmatrix}64\\64\\-64\\-64\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}32\\-32\\0\\0\\0\\0\\0\\0\end{bmatrix} - \begin{bmatrix}0\\0\\0\\0\\16\\-16\\0\\0\end{bmatrix} = \begin{bmatrix}96\\32\\192\\192\\112\\144\\128\\128\end{bmatrix}$$

This is a grayscale image with the first pixel at intensity of 96, the second pixel at intensity of 32, the third pixel at intensity of 192, the fourth pixel at intensity of 192, the fifth pixel at intensity of 112, the sixth pixel at intensity of 144, the seventh pixel at intensity of 128, and the eighth pixel at intensity of 128.

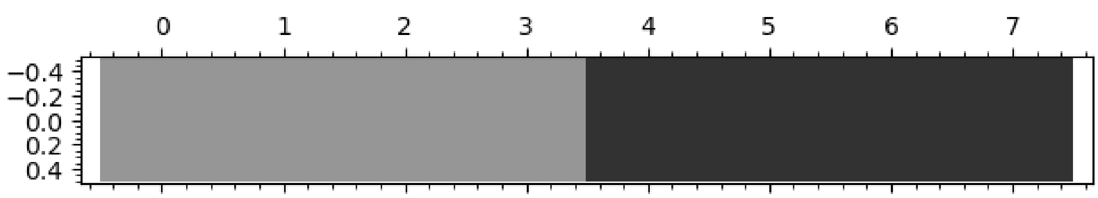
See the sage codecell below for the images.

[4]:
```
# combination 1
combination_1 = v1
print(f"The combination 1 is:\n{combination_1}")
show_image(combination_1)

# combination 2
combination_2 = 100*v1 + 50*v2
print(f"The combination 2 is:\n{combination_2}")
show_image(combination_2)

# combination 3
combination_3 = 128*v1 - 64*v3 + 32*v5 - 16*v7
print(f"The combination 3 is:\n{combination_3}")
show_image(combination_3)
```

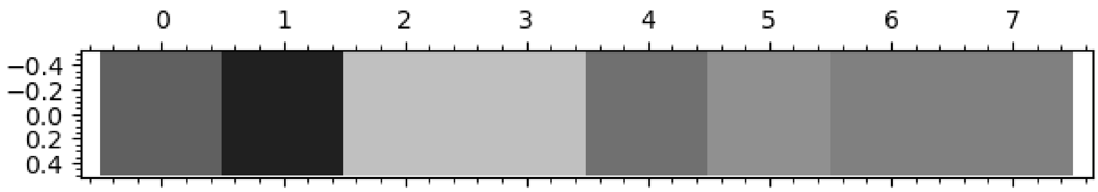
```
The combination 1 is:
(1, 1, 1, 1, 1, 1, 1, 1)
```




```
The combination 2 is:
(150, 150, 150, 150, 50, 50, 50, 50)
```




```
The combination 3 is:
(96, 32, 192, 192, 112, 144, 128, 128)
```



In general, v_1 is used to increase the intensity of all the pixels. This is the case because v_1 is a vector of all 1's. As such, adding v_1 to a vector will increase the intensity of all the pixels by the corresponding coefficient. For example, adding $100v_1$ to a vector will increase the intensity of all the pixels by 100.

The vector v_2 is used to increase the intensity of the first four pixels and decrease the intensity of the last four pixels. This is the case because the first four pixels are 1's and the last four pixels are -1's. As such, adding $50v_2$ to a vector will increase the intensity of the first four pixels by 50 and decrease the intensity of the last four pixels by 50.

The vector v_3 is used to increase the intensity of the first two pixels and decrease the intensity of the last two pixels. This is the case because the first two pixels are 1's and the last two pixels are -1's. As such, adding $-64v_3$ to a vector will increase the intensity of the first two pixels by 64 and decrease the intensity of the last two pixels by 64.

The vector v_4 performs the same operation as v_3 but it does this on the last 4 pixels of the image. This is the case because the the first four pixels are 0's and the next two are 1's and the last two are -1's.

vectors v_5, v_6, v_7, v_8 brighten and darken the pixels in the same way as v_3 and v_4 but they do this on two pixels only.

In general, linear combinations of the vectors $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$ can be used to strategically intensify or dim the pixels of an image. This will allow use to create any vector in $\mathbb{R}^8$ which is the coefficients of the linear combination of the standard basis vectors. As we disussed in (1), the coordinates of the vector are the intensities of the pixels. As such, we can create any image we want by using linear combinations of the vectors $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$.

0.0.4 3

(a)

To find the coordinates of v in the new basis, we can use the formula

$$v = \sum_{i=1}^{8} c_i v_i$$

where v_i are the vectors in the new basis and c_i are the coordinates of v in the new basis.

$$v = c_1\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\\1\end{bmatrix} + c_2\begin{bmatrix}1\\1\\1\\1\\-1\\-1\\-1\\-1\end{bmatrix} + c_3\begin{bmatrix}1\\1\\-1\\-1\\0\\0\\0\\0\end{bmatrix} + c_4\begin{bmatrix}0\\0\\0\\0\\1\\1\\-1\\-1\end{bmatrix} + c_5\begin{bmatrix}1\\-1\\0\\0\\0\\0\\0\\0\end{bmatrix} + c_6\begin{bmatrix}0\\0\\1\\-1\\0\\0\\0\\0\end{bmatrix} + c_7\begin{bmatrix}0\\0\\0\\0\\1\\-1\\0\\0\end{bmatrix} + c_8\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\-1\end{bmatrix}$$

We can rewrite this relation in the form $Ac = v$ where A is the matrix whose columns are the vectors $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$ and c is the vector whose coordinates are the coordinates of v in the new basis.

$$\begin{bmatrix}1 & 1 & 1 & 0 & 1 & 0 & 0 & 0\\ 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0\\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0\\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0\\ 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0\\ 1 & -1 & 0 & 1 & 0 & 0 & -1 & 0\\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 1\\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & -1\end{bmatrix}\begin{bmatrix}c_1\\c_2\\c_3\\c_4\\c_5\\c_6\\c_7\\c_8\end{bmatrix} = \begin{bmatrix}31\\159\\9\\162\\233\\54\\217\\3\end{bmatrix}$$

We can solve this system of equations to find the coordinates of v in the new basis.

$$\begin{bmatrix}c_1\\c_2\\c_3\\c_4\\c_5\\c_6\\c_7\\c_8\end{bmatrix} = \begin{bmatrix}1 & 1 & 1 & 0 & 1 & 0 & 0 & 0\\ 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0\\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0\\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0\\ 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0\\ 1 & -1 & 0 & 1 & 0 & 0 & -1 & 0\\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 1\\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & -1\end{bmatrix}^{-1}\begin{bmatrix}31\\159\\9\\162\\233\\54\\217\\3\end{bmatrix}$$

See sage code below for the calculation.

$$\begin{bmatrix}c_1\\c_2\\c_3\\c_4\\c_5\\c_6\\c_7\\c_8\end{bmatrix} = \begin{bmatrix}217/2\\-73/4\\19/4\\67/4\\-64\\-153/2\\179/2\\107\end{bmatrix}$$

[5]:
```
A_inverse = A.inverse()
v = vector([31,159,9,162,233,54,217,3])
```

```
# finding the coordinated of v in the new basis
new_coordinates = A_inverse*v
print(f'The coordinates of v in the new basis are:\n{new_coordinates}')
```

```
The coordinates of v in the new basis are:
(217/2, -73/4, 19/4, 67/4, -64, -153/2, 179/2, 107)
```

(b)

Remember that the coordinates of v in the new basis is

$$c = \begin{bmatrix} 217/2 \\ -73/4 \\ 19/4 \\ 67/4 \\ -64 \\ -153/2 \\ 179/2 \\ 107 \end{bmatrix}$$

let the first threshold $\epsilon_1 = 0$

With the first threshold, we set all values in c for which $|c_i|$ that are less than ϵ_1 to 0. This means that we set none of the coordinates to 0 because none of the coordinates have their absolute values less than 0. The new vector c_1 is

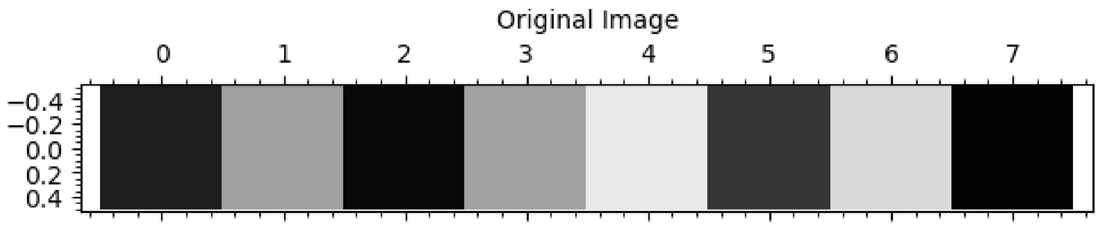
$$c_1 = \begin{bmatrix} 217/2 \\ -73/4 \\ 19/4 \\ 67/4 \\ -64 \\ -153/2 \\ 179/2 \\ 107 \end{bmatrix}$$

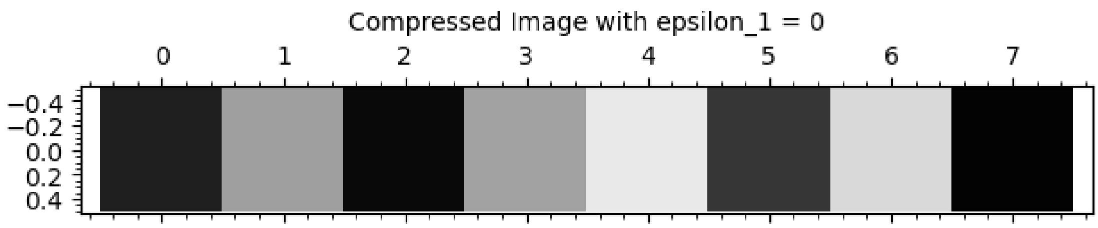
in order to represent c_1 in the standard basis, we will multiply c_1 by A where A is the matrix whose columns are the vectors $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 217/2 \\ -73/4 \\ 19/4 \\ 67/4 \\ -64 \\ -153/2 \\ 179/2 \\ 107 \end{bmatrix} = \begin{bmatrix} 31 \\ 159 \\ 9 \\ 162 \\ 233 \\ 54 \\ 217 \\ 3 \end{bmatrix}$$

As we can see, the vector c_1 in our new basis is the same as the vector v in the standard basis. As such the pixels in the original image and the compressed image have the same pixel intensities when epsilon is 0. We can say that no compression has occurred. (see sage code below for both images)

[6]:
```
epsilon_1 = 0
original = vector([31,159,9,162,233,54,217,3])

# function to find the new coordinates of the vector v in the new basis given an epsilon value
def find_new_coordinates(vec,epsilon):
    '''
    Description
    -----------
    Returns the new coordinates of the vector v in the new basis given an epsilon value.
    it removes the coordinates whose absolute value is less than epsilon.

    Parameters
    ----------
    vec : vector
        The vector v.
    epsilon : float
        The epsilon value.

    Returns
    -------
    vector
    '''
    # storage vector to store the new coordinates
    storage_vector = []
    for pixel in vec:
        # if the absolute value of the coordinate is less than epsilon, then the coordinate is removed
        if abs(pixel) < epsilon:
            storage_vector.append(0)
        else:
            storage_vector.append(pixel)
    return vector(storage_vector)

# c_1 given epsilon_1 = 0
c_1 = find_new_coordinates(new_coordinates,epsilon_1)

# plotting the original image and the compressed image with epsilon_1 = 0
show_image(original, title="Original Image")
show_image(A*c_1, title="Compressed Image with epsilon_1 = 0")
```





let the second threshold $\epsilon_2 = 10$

Performing the same steps we did for ϵ_1, the new vector c_2 is

$$c_2 = \begin{bmatrix} 217/2 \\ -73/4 \\ 0 \\ 67/4 \\ -64 \\ -153/2 \\ 179/2 \\ 107 \end{bmatrix}$$

We can represent c_2 in the standard basis by multiplying c_2 by A where A is the matrix whose columns are the vectors $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$.

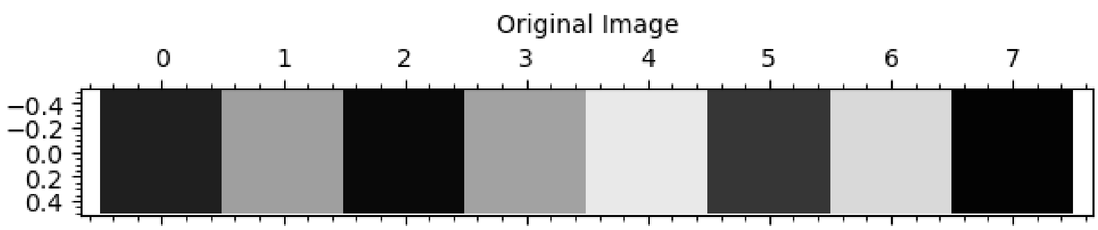
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 217/2 \\ -73/4 \\ 0 \\ 67/4 \\ -64 \\ -153/2 \\ 179/2 \\ 107 \end{bmatrix} = \begin{bmatrix} 26.25 \\ 154.25 \\ 13.75 \\ 166.75 \\ 233 \\ 54 \\ 217 \\ 3 \end{bmatrix}$$

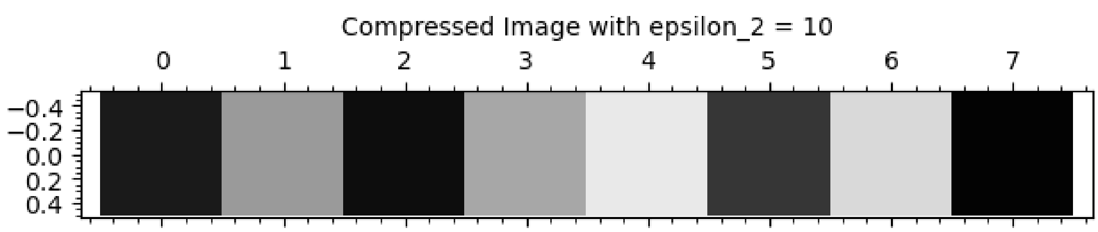
The vector c_2 in the standard basis is not the same as the vector v. This is a new vector that is different from the original vector v. As such, the image formed is different from the original image.

However, the image formed is still very similar to the original image to the human eye. (see sage code below for both images)

[7]:
```
epsilon_2 = 10
c_2 = find_new_coordinates(new_coordinates,epsilon_2)
print(f"The coordinates of v in the new basis with epsilon_2 = 10 are:\n{c_2}")

# plotting the original image and the compressed image with epsilon_2 = 10
show_image(original, title="Original Image")
show_image(A*c_2, title="Compressed Image with epsilon_2 = 10")
```

```
The coordinates of v in the new basis with epsilon_2 = 10 are:
(217/2, -73/4, 0, 67/4, -64, -153/2, 179/2, 107)
```





Let the third threshold $\epsilon_3 = 50$

Performing the same steps we did for ϵ_2, the new vector c_3 is

$$c_3 = \begin{bmatrix} 217/2 \\ 0 \\ 0 \\ 0 \\ -64 \\ -153/2 \\ 179/2 \\ 107 \end{bmatrix}$$

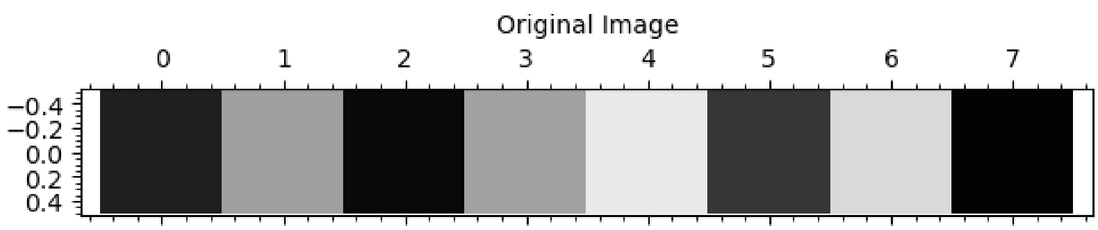
We can represent c_3 in the standard basis by multiplying c_3 by A where A is the matrix whose columns are the vectors $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$.

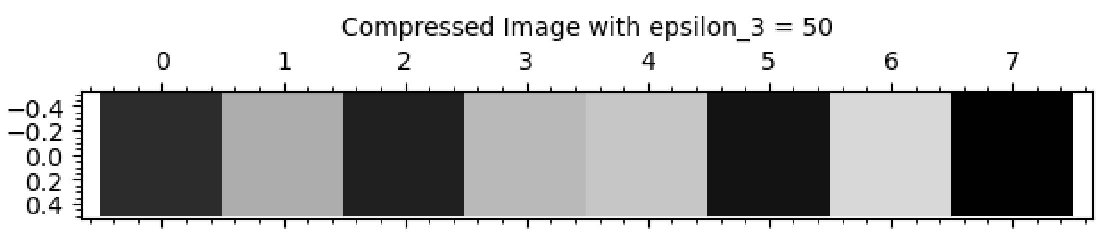
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 217/2 \\ 0 \\ 0 \\ 0 \\ -64 \\ -153/2 \\ 179/2 \\ 107 \end{bmatrix} = \begin{bmatrix} 44.5 \\ 172.5 \\ 32 \\ 185 \\ 198 \\ 19 \\ 215 \\ 1.5 \end{bmatrix}$$

The vector c_3 in the standard basis is not the same as the vector v. This is a new vector that is different from the original vector v. As such, the image formed is different from the original image. However, the image formed is still very similar to the original image to the human eye. (see sage code below for both images)

[8]:
```
epsilon_3 = 50
c_2 = find_new_coordinates(new_coordinates,epsilon_3)
print(f"The coordinates of v in the new basis with epsilon_3 = 50 are:\n{c_2}")

# plotting the original image and the compressed image with epsilon_3 = 50
show_image(original, title="Original Image")
show_image(A*c_2, title="Compressed Image with epsilon_3 = 50")
```

```
The coordinates of v in the new basis with epsilon_3 = 50 are:
(217/2, 0, 0, 0, -64, -153/2, 179/2, 107)
```





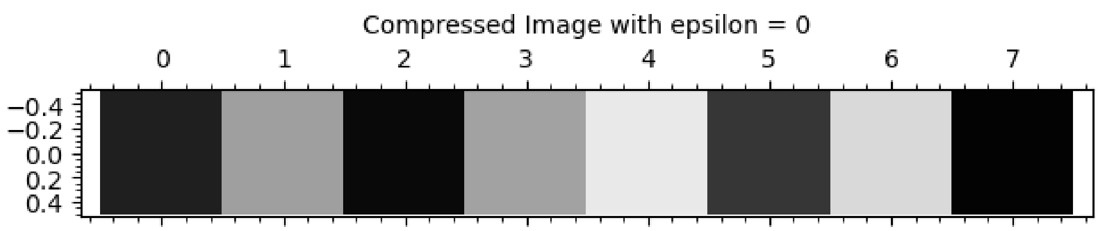
The vectors $c_1, c_2,$ and c_3 in the standard basis all represent the image formed after compressing the vector v in our new basis using different thresholds. All images formed look very similar (to the human eye) to the original image formed from v.

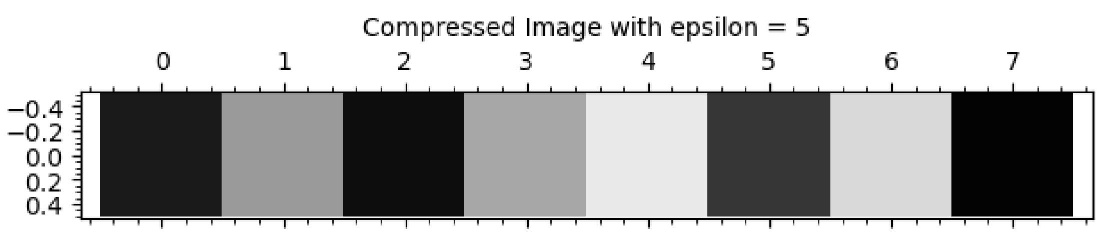
(c)

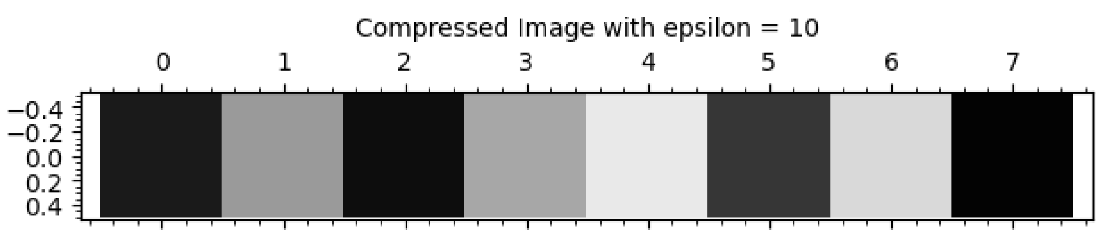
To determine the value of ϵ for which we will start to notice the effect of the compression, we will test out different values of ϵ and see what each image looks like. See the codecell below for the code that will generate the images.

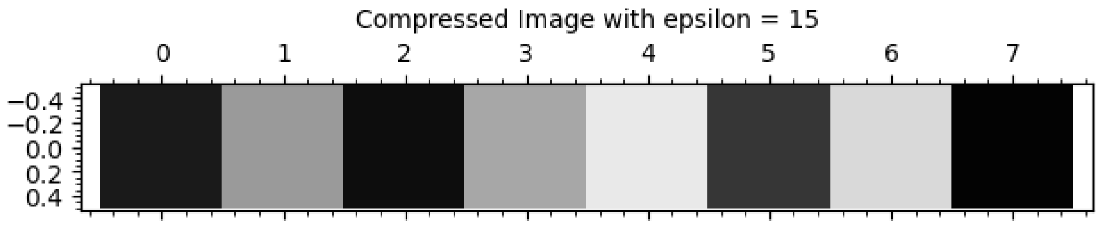
Looking at the images below, at around $\epsilon = 65$ we start to notice the effect of the compression as the first two pixels seem to merge to become one grey pixel. As epsilon increases the effect becomes even more noticable as more pixels start to merge together to become one grey pixel.

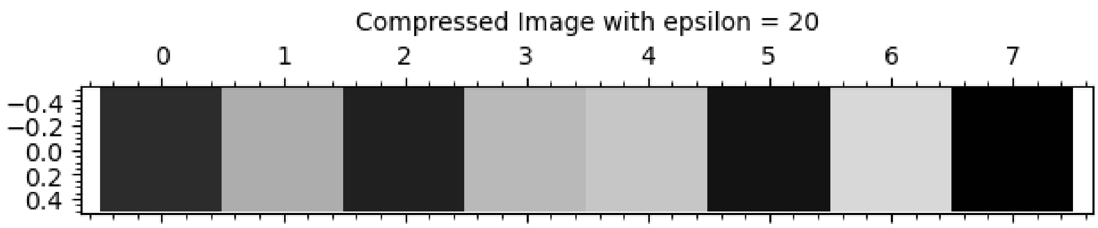
[9]:
```
# value of epsilon to start with, the value of epsilon to end with, and the
↪step size
start_val = 0
end_val = 90
step = 5

# Visualizing the image formed by compressing the original image with different
↪values of epsilon
for epsilon in range(start_val,end_val,step):
    c = find_new_coordinates(new_coordinates,epsilon)
    show_image(A*c, title=f"Compressed Image with epsilon = {epsilon}")
```

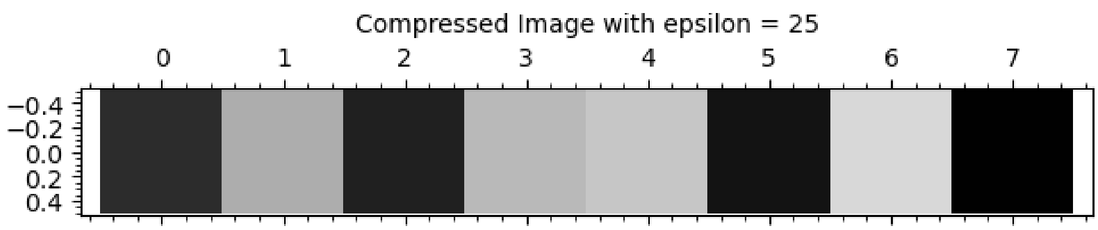


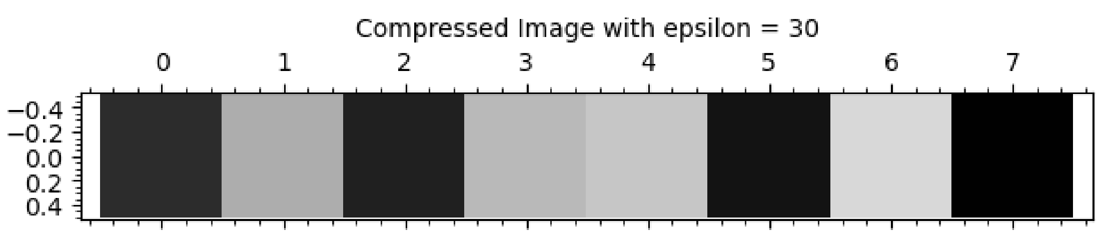


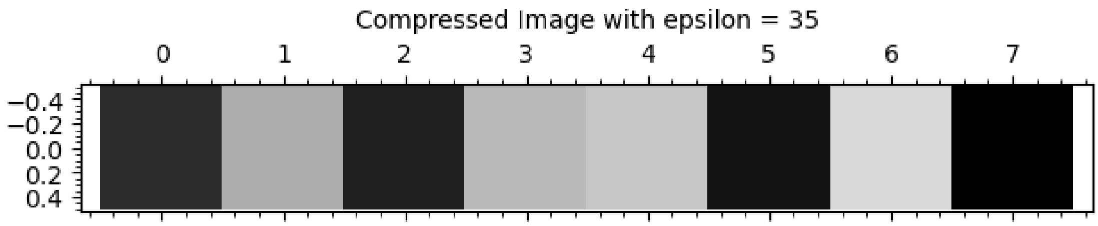


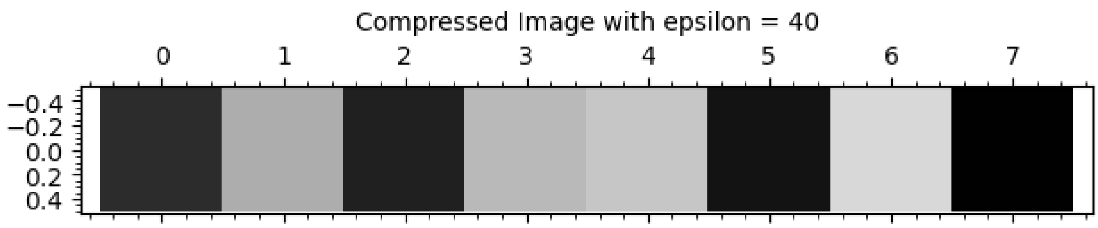

Compressed Image with epsilon = 15
0
1
2
3
4
5
6
7
−0.4
−0.2
0.0
0.2
0.4

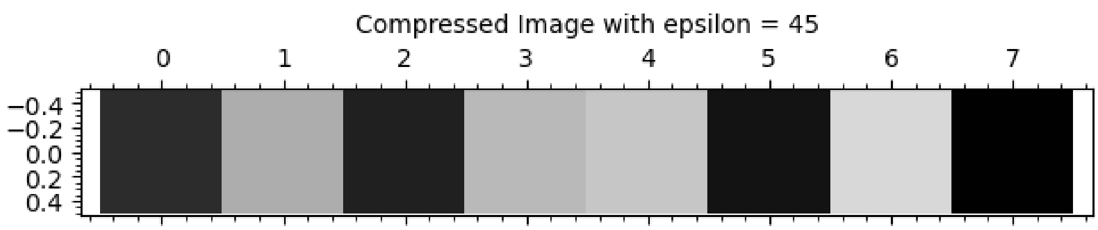

Compressed Image with epsilon = 20
0
1
2
3
4
5
6
7
−0.4
−0.2
0.0
0.2
0.4

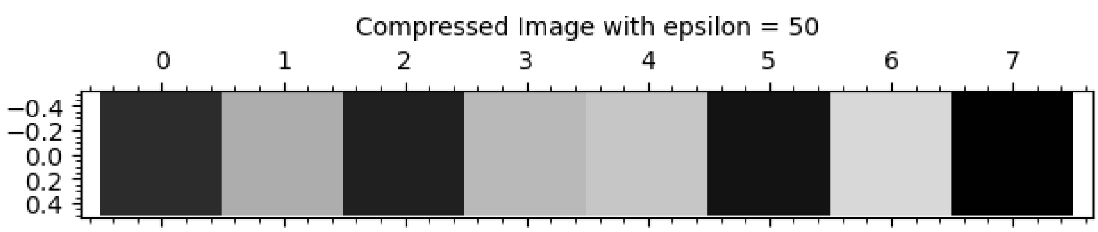

Compressed Image with epsilon = 25
0
1
2
3
4
5
6
7
−0.4
−0.2
0.0
0.2
0.4

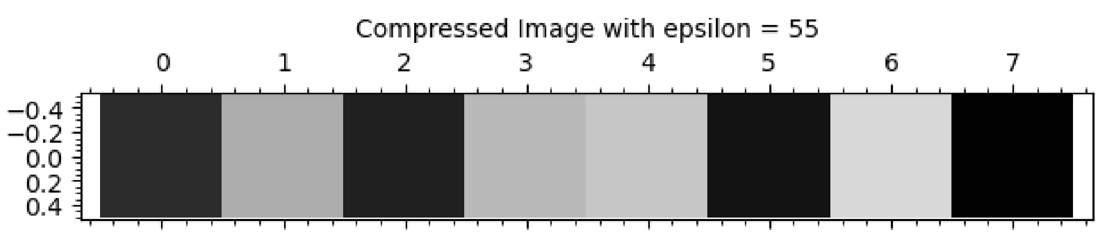

Compressed Image with epsilon = 30
0
1
2
3
4
5
6
7
−0.4
−0.2
0.0
0.2
0.4

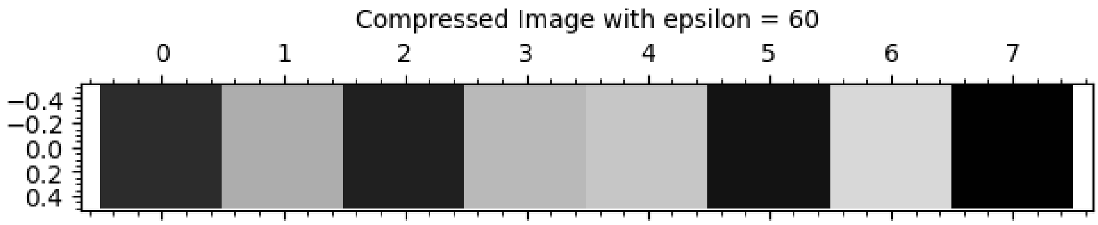

Compressed Image with epsilon = 35
0
1
2
3
4
5
6
7
−0.4
−0.2
0.0
0.2
0.4

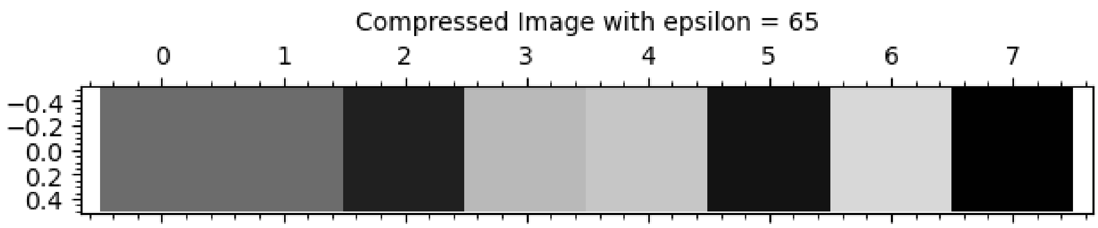

Compressed Image with epsilon = 40
0
1
2
3
4
5
6
7
−0.4
−0.2
0.0
0.2
0.4

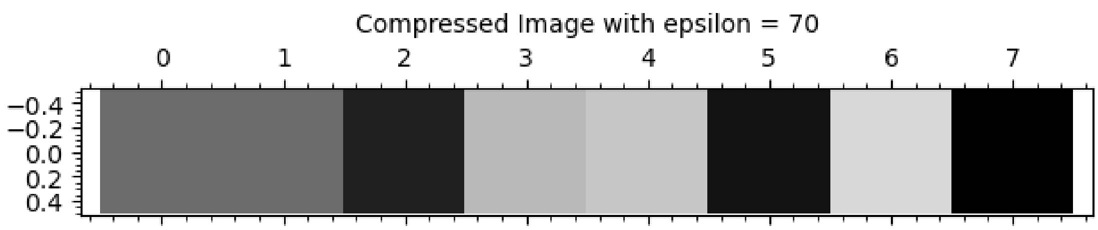

Compressed Image with epsilon = 45
0
1
2
3
4
5
6
7
−0.4
−0.2
0.0
0.2
0.4

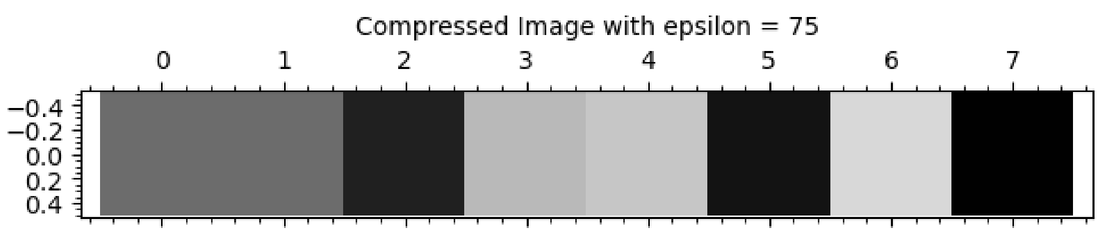

Compressed Image with epsilon = 50
0
1
2
3
4
5
6
7
−0.4
−0.2
0.0
0.2
0.4

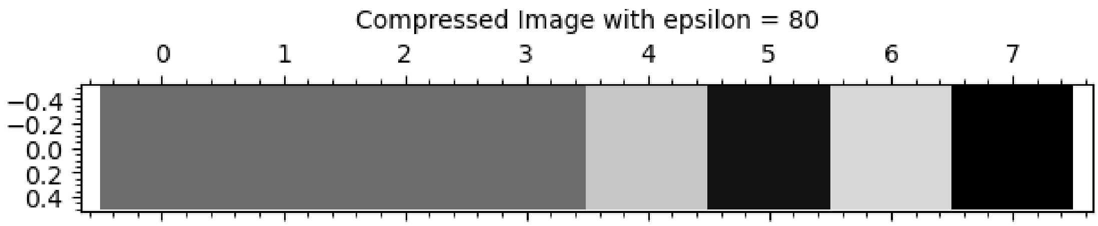

Compressed Image with epsilon = 55
0
1
2
3
4
5
6
7
−0.4
−0.2
0.0
0.2
0.4

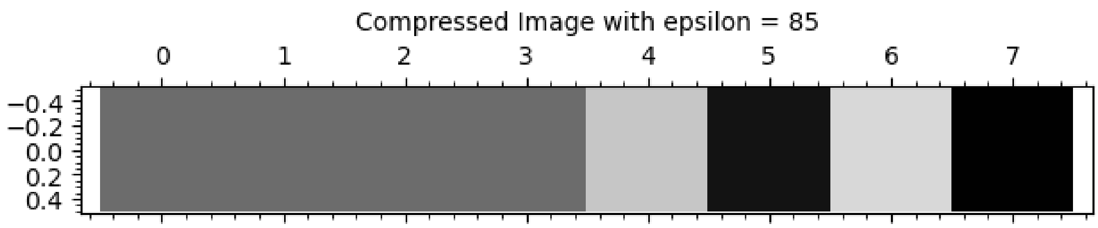

Compressed Image with epsilon = 60
0 1 2 3 4 5 6 7
−0.4 −0.2 0.0 0.2 0.4


Compressed Image with epsilon = 65
0 1 2 3 4 5 6 7
−0.4 −0.2 0.0 0.2 0.4


Compressed Image with epsilon = 70
0 1 2 3 4 5 6 7
−0.4 −0.2 0.0 0.2 0.4


Compressed Image with epsilon = 75
0 1 2 3 4 5 6 7
−0.4 −0.2 0.0 0.2 0.4





0.0.5 4

We can extend the above method for a matrix with 64 pixels (an 8×8 matrix). Given an 8×8 matrix X whose column vectors are in the standard basis (in $\mathbb{R}^8$). Let X be given by

$$X = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ col_1 & col_2 & col_3 & col_4 & col_5 & col_6 & col_7 & col_8 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

If we compress this matrix, we can still apply the same methods in **(3)** above. We will start by changing the matrix to the new basis. We will do this by multiplying X by A where A^{-1} is the matrix whose columns are the vectors $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$ (the basis vectors of our new basis). The matrix Q formed will contain the coordinates of the column vectors of X in the new basis. Q will be given by

$$Q = A^{-1}X$$

We will then apply the same thresholding method as in **(3)** above. We will start by choosing a threshold ϵ. We will then set all the coordinates of Q that are less than ϵ to be 0. We will then multiply Q by A to get the matrix Y whose column vectors are in the standard basis. Y will be given by

$$Y = AQ$$

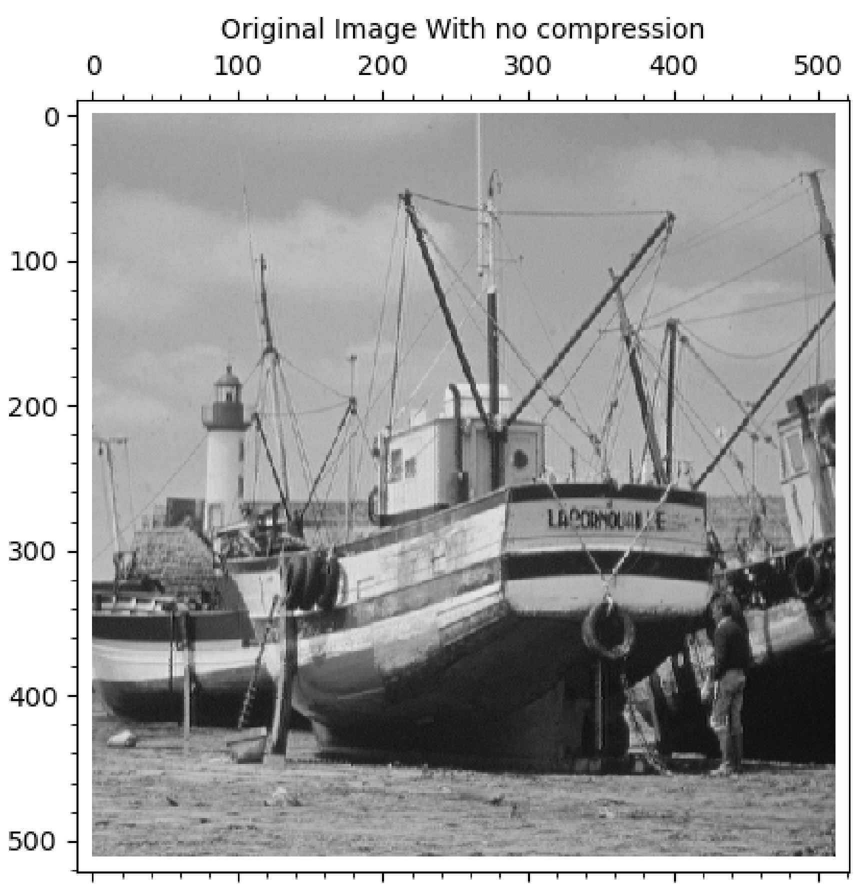
The matrix Y will be the image formed after compressing the matrix X in our new basis using the threshold ϵ.

Image sizes are usually in multiples of 8. For example, a 512×512 image is made up of 64 (512/8) 8×8 matrices. If we compress each of these 8×8 matrices using the method above, we will get a compressed image. The compressed image will be made up of 64 8×8 compressed images. The compressed image will be a 512×512 matrix (The same number of pixels as the original image).

As such, inorder to compress an image, we will need to break the image into 8×8 matrices and compress each of these matrices using the method above. We will then put the compressed matrices back together to form the compressed image.

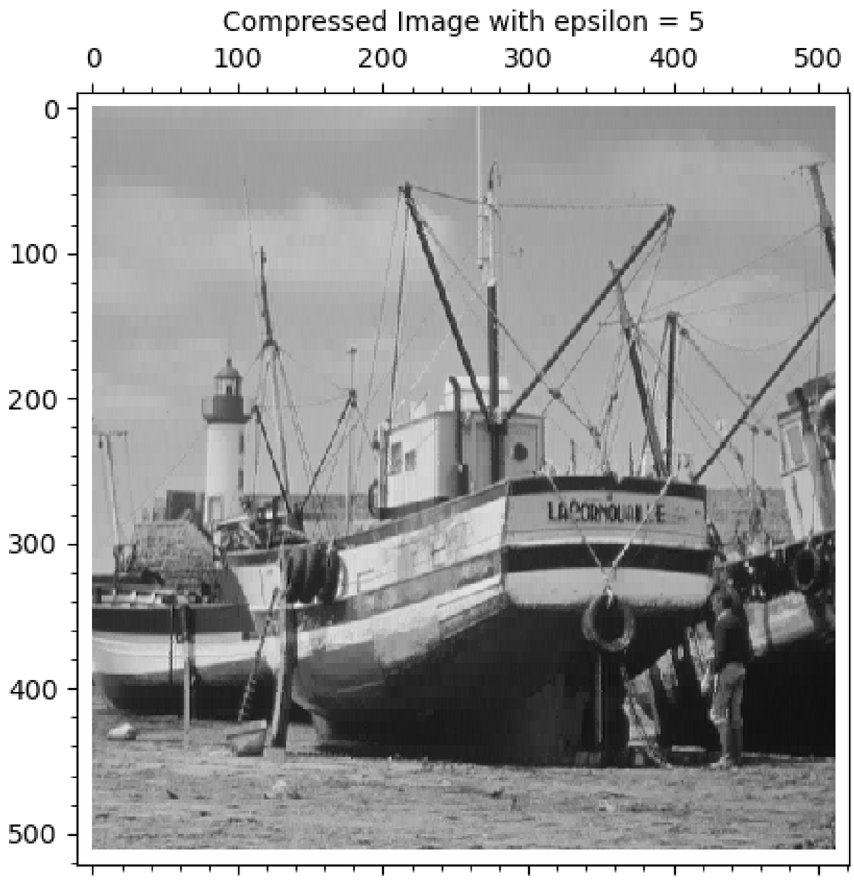
Below we will demonstrate the compression of an image using the method above.

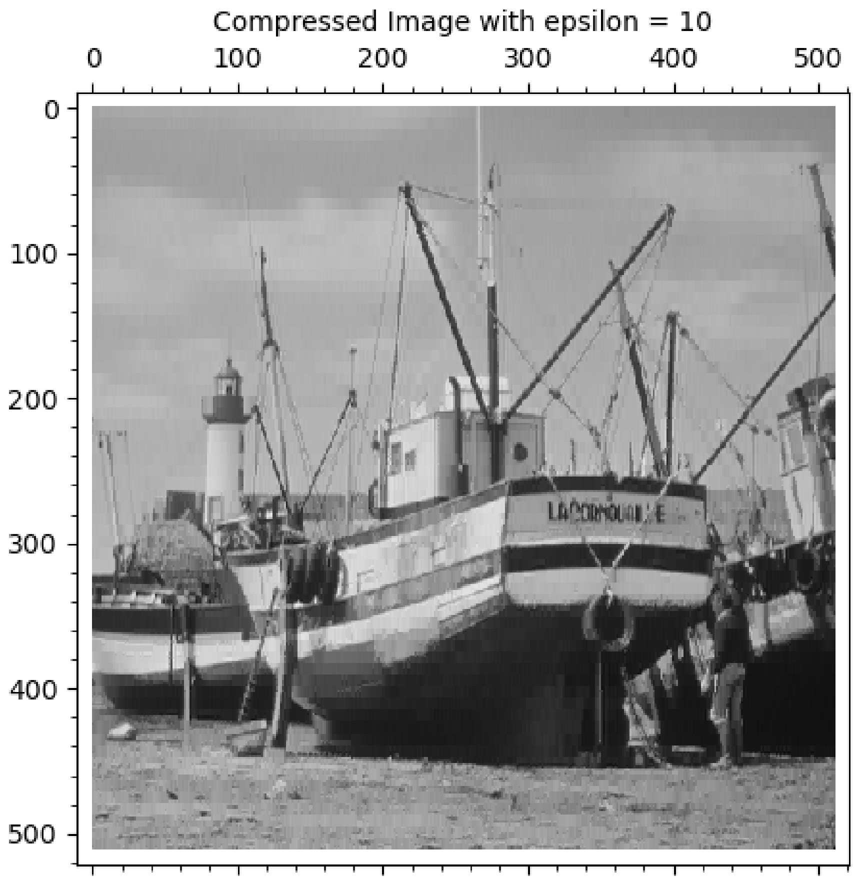
[65]:
```python
#loading the image
from PIL import Image
import numpy as np
img = Image.open("boat.tif")
img = np.array(img)
pixel_matrix = matrix(QQ,img)

# displaying the image
show_image(pixel_matrix, title="Original Image With no compression")
```

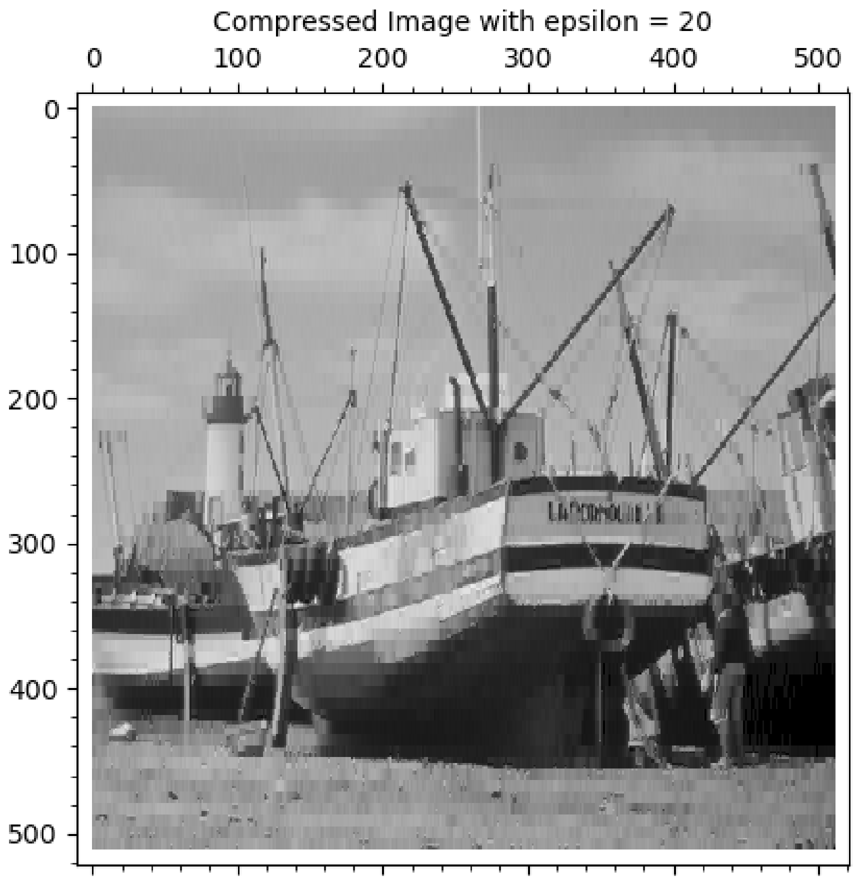


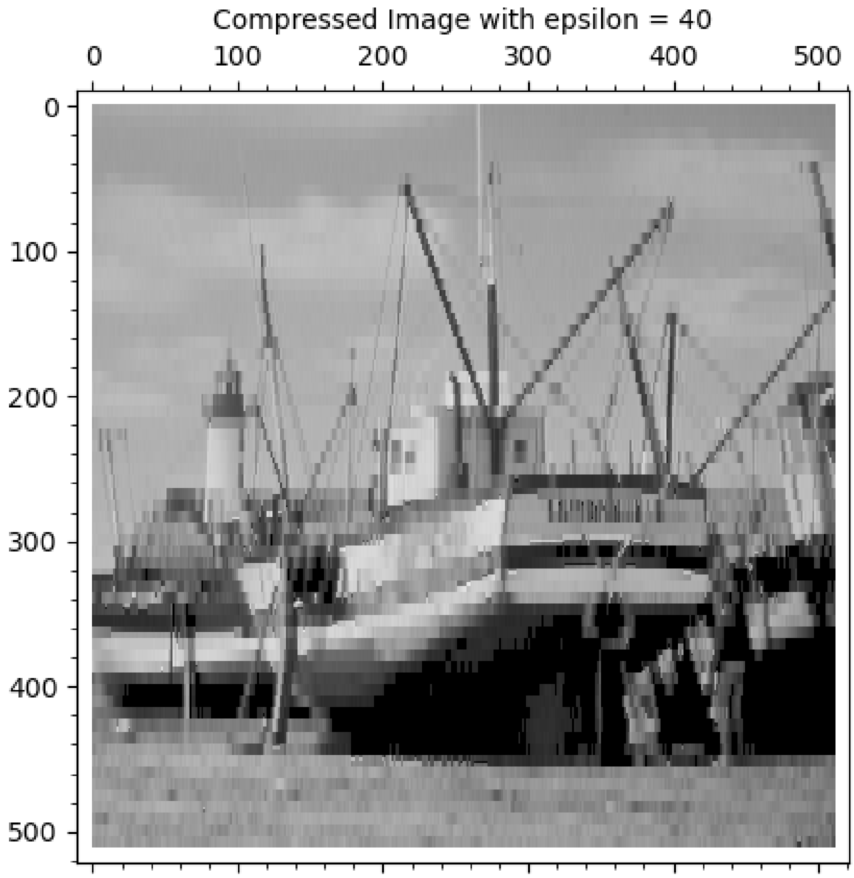
[42]:
```
# image compression function
def compress_image(img_matrix, epsilon):
    '''
    Description
    -----------
    Returns the compressed image given a matrix of pixel values and an epsilon␣
 ↪value.

    Parameters
    ----------
    matrix : matrix
        A matrix of pixel values.
    epsilon : float
        The epsilon value.

    Returns
    -------
    matrix
    '''
    row_length = img_matrix.nrows()
    col_length = img_matrix.ncols()

    # breaking the matrix into 8x8 blocks
    for i in range(0,row_length,8):
        for j in range(0,col_length,8):

            # the block of pixels to be compressed
            block = img_matrix[i: i+8, j: j+8]

            # the block in the new basis
            block_in_new_basis = A.inverse() * block

            # temporary storage matrix to store the compressed block
            temp = []
            for row in block_in_new_basis:
                temp.append(find_new_coordinates(row,epsilon))
            temp = matrix(temp)

            # the compressed block in the original basis
            compressed_block = A*temp

            # replacing the block of pixels with the compressed block                   ␣
 ↪
            img_matrix[i:i+8,j:j+8] = compressed_block

    return img_matrix
```

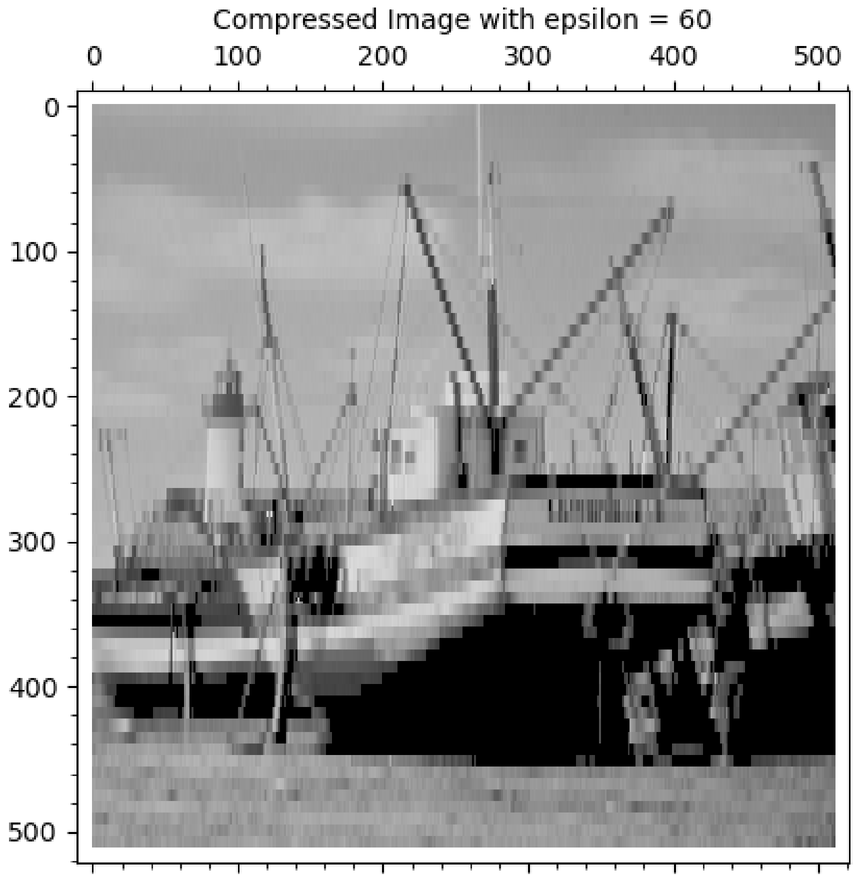
[66]:
```
epsilon_values = [5, 10, 20, 40, 60]

for epsilon in epsilon_values:
    pixel_matrix = matrix(QQ,img)
    new = compress_image(pixel_matrix,epsilon)
    show_image(new, title=f"Compressed Image with epsilon = {epsilon}")
```




Compressed Image with epsilon = 10
0
100
200
300
400
500
LA CORNOUAILLE


Compressed Image with epsilon = 20
0
100
200
300
400
500

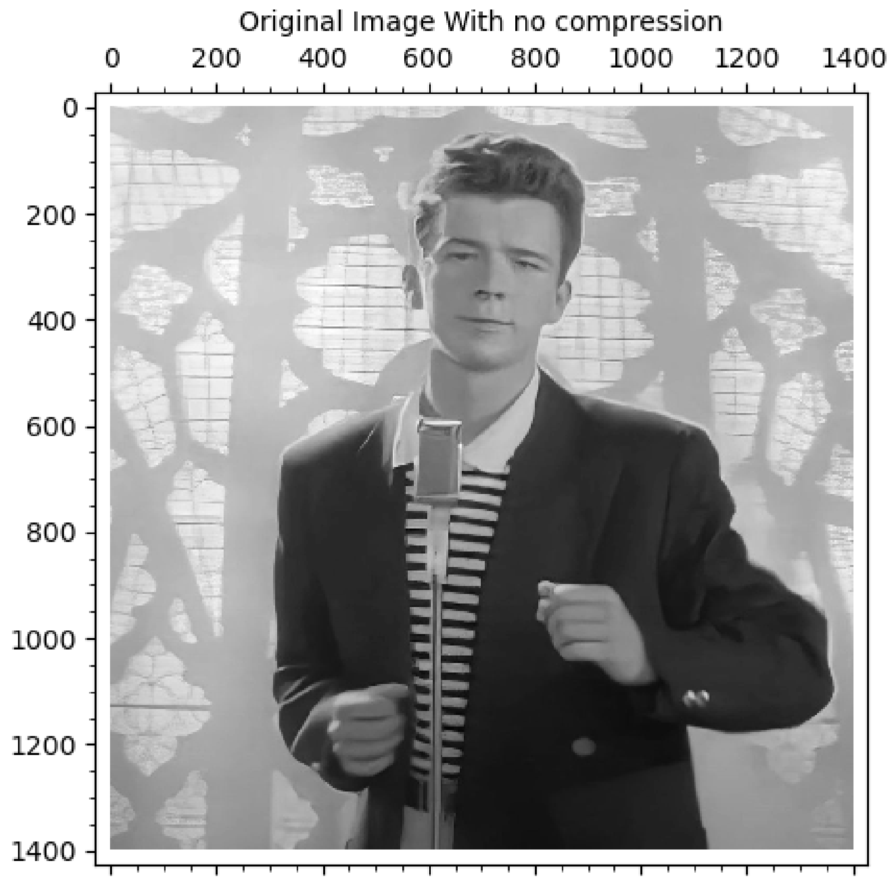

Compressed Image with epsilon = 40
0
100
200
300
400
500
0
100
200
300
400
500



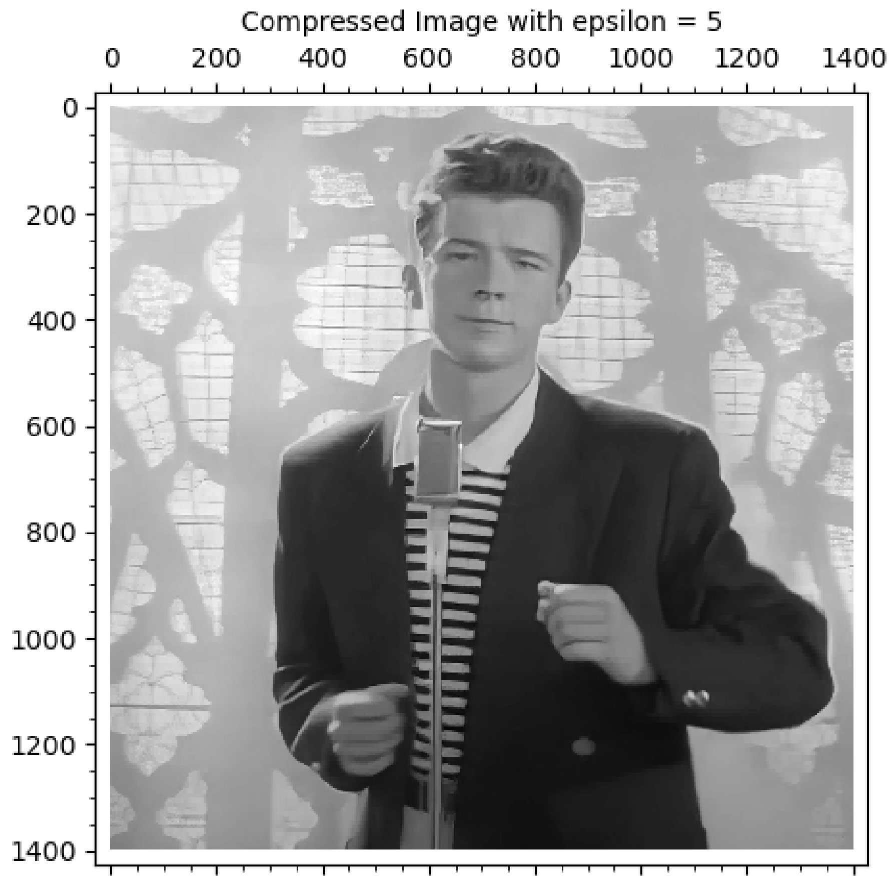
Above, we have a 512 × 512 image being compressed by following the method above. The first image is the original image. As we increase the threshold ϵ, we start to notice the effect of the compression as the image starts to loose more and more detail.

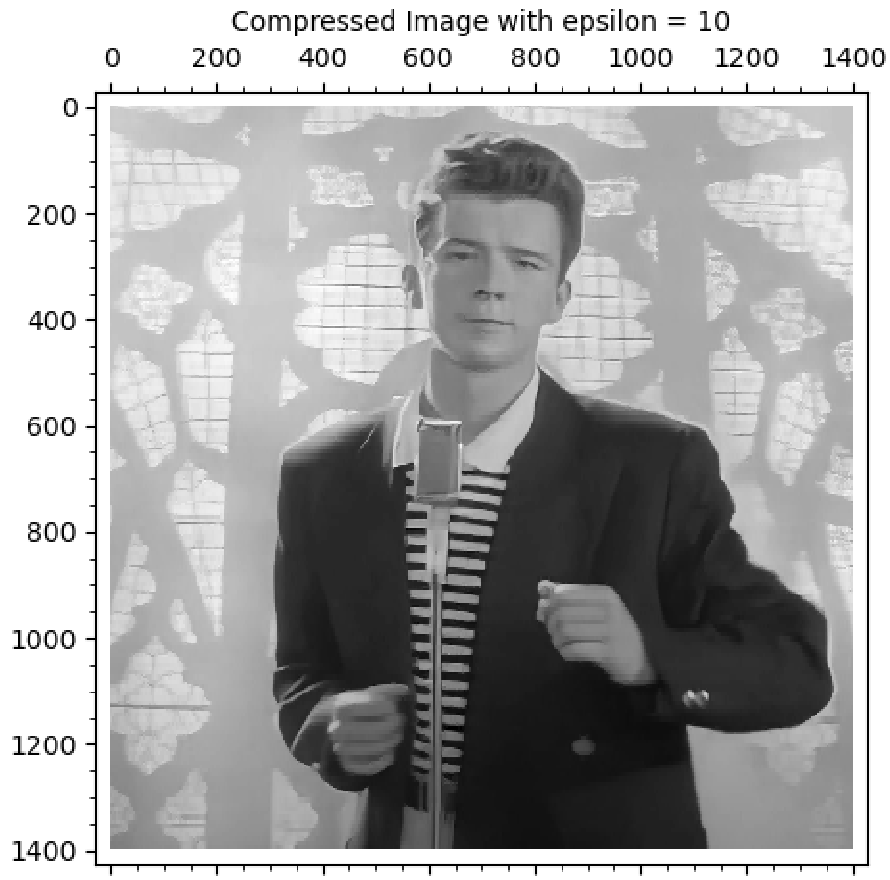
Additional test of the compression function

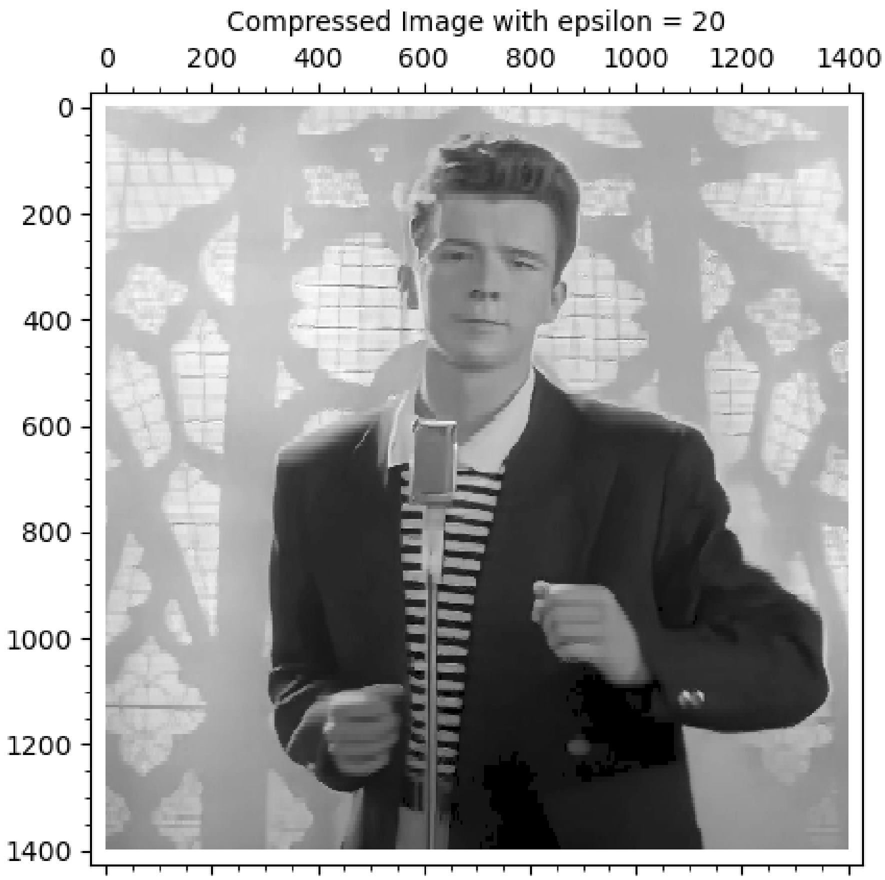
[67]:
```
#loading the image
from PIL import Image
import numpy as np
img = Image.open("rickroll.tif")
img = np.array(img)
pixel_matrix = matrix(QQ,img)

# displaying the image
show_image(pixel_matrix, title="Original Image With no compression")
```

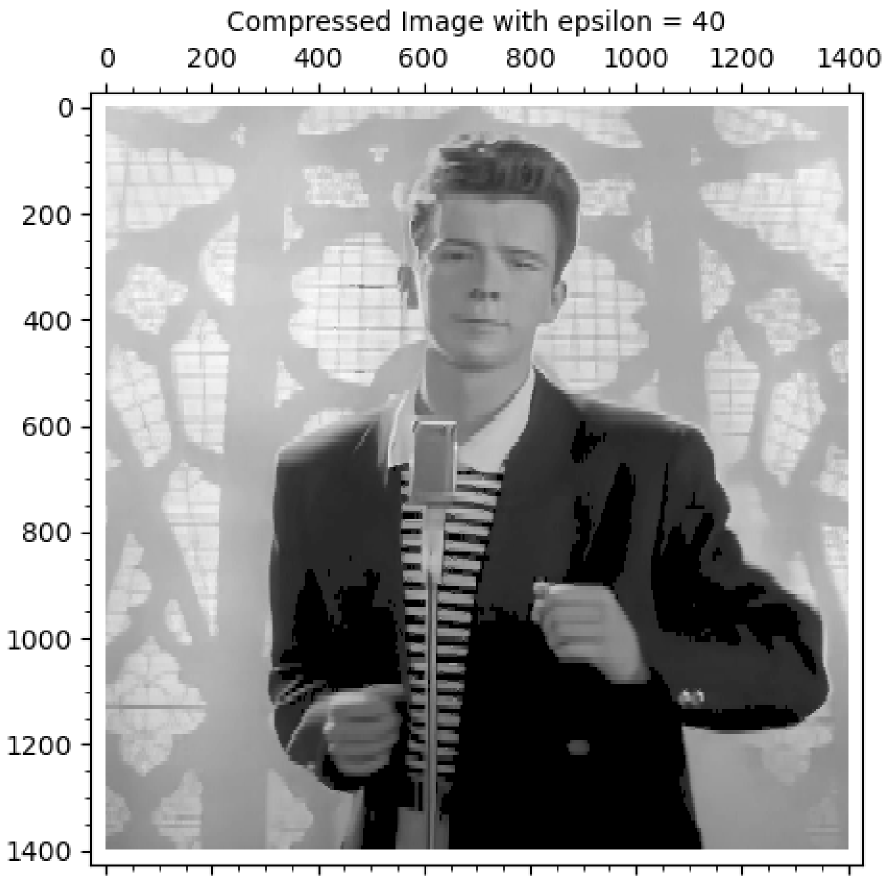


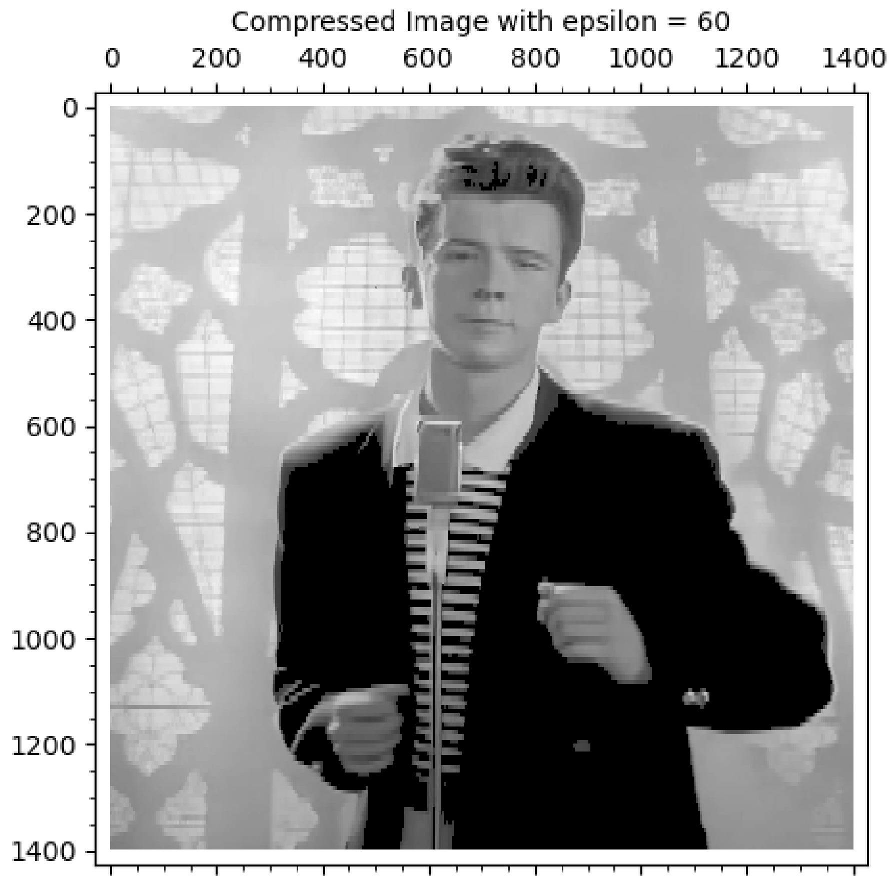
[68]:
```
epsilon_values = [5, 10, 20, 40, 60]

for epsilon in epsilon_values:
    pixel_matrix = matrix(QQ,img)
    new = compress_image(pixel_matrix,epsilon)
    show_image(new, title=f"Compressed Image with epsilon = {epsilon}")
```


Compressed Image with epsilon = 5
0
200
400
600
800
1000
1200
1400


Compressed Image with epsilon = 10
0
200
400
600
800
1000
1200
1400
0
200
400
600
800
1000
1200
1400


Compressed Image with epsilon = 20
0
200
400
600
800
1000
1200
1400
0
200
400
600
800
1000
1200
1400


Compressed Image with epsilon = 40
0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400



0.0.6 Reflection

(1)

(a)

#breakitdown: I applied this HC in problem 4. The new basis that we are converting to from the standard basis in in $\mathbb{R}^8$. As such, we cannot easily apply the compression on a 512×512 image. This is because the matrix X will be a 512×512 matrix (this is not in $\mathbb{R}^8$). As such, I solved the problem by breaking the image into 8×8 matrices and then applying the compression method on these matrices. I then put the compressed matrices back in their original positions to form the compressed image. This is an effective use of **#breakitdown** because I was able to breakdown the problem into smaller sub problems and tackle each of these sub problems individually. This made the problem easier to solve.

(b)

#Algorithms: I applied this HC throughout this assignment. The goal of the assignment is to compress an image. As such, I had to come up with an algorithm to compress the image. The input of the algorithm is a matrix of pixel intensities and ϵ (the threshold of compression). The output of the algorithm is a new matrix that whose values are the pixel intensities of the compressed image.

The algorithm is as follows:

1. Break the image into 8×8 matrices.
2. for each 8×8 matrix, convert the matrix to the new basis.
3. set all the coordinates of the matrix that are less than ϵ to be 0.
4. return the matrix to the standard basis.
5. combine all the compressed 8×8 matrices back into the compressed image.

This is an effective use of **#Algorithms** because the algorithm has a well defined input, output, and steps. This makes it easy to implement the algorithm. Additionally, I was able to write this algorithm in code.

In order to improve my application of this algorithm, I would need to talk about how efficient the algorithm is at compressing the image.

(2)

I found change of basis particularlly interesting. I found it interesting to see how the change of basis can be used to compress a grayscale image. This is because it is easy to change a matrix to a new basis. Changing the pixel matrix to a new basis and dropping the coordinates that are less than ϵ is a very simple process. As such, it is easy to compress an image using this method. It seems counterintuitive that dropping some coordinates will not damage up the image significantly. However, it is interesting to see that this is the case. I think that is the power of the specific basis that we used. It is interesting to see why that particular basis of $\mathbb{R}^8$ is the best basis to use for compressing a grayscale image. After some reserch, I found the name of the basis that we used in this assignment. The basis that we used is called the Haar Wavelet Basis.