PROBLEM 1

① $v_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \longrightarrow \boxed{u_1 = \frac{v_1}{\|v_1\|} = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}}$

② $\tilde{u}_2 = v_2 - \underbrace{P_{\text{span}(u_1)}(v_2)}$

$\langle v_2, u_1 \rangle u_1 = \left(2/\sqrt{3} + 1/\sqrt{3} + 1/\sqrt{3} \right) u_1$

$= 4/\sqrt{3} \begin{pmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{pmatrix} = 4/3 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

$\Rightarrow \tilde{u}_2 = \begin{pmatrix} 2 - 4/3 \\ 1 - 4/3 \\ 1 - 4/3 \end{pmatrix} = \begin{pmatrix} 2/3 \\ -1/3 \\ -1/3 \end{pmatrix}$

and $\|\tilde{u}_2\|_2 = \sqrt{(2/3)^2 + (1/3)^2 + (1/3)^2} = \frac{1}{3}\sqrt{6}$

$\rightarrow u_2 = \frac{\tilde{u}_2}{\|\tilde{u}_2\|_2} = \frac{1}{\sqrt{6}} \begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix}$

③ $\tilde{u}_3 = v_3 - \underbrace{P_{\text{span}(u_1, u_2)}(v_3)}$

$= \langle v_3, u_1 \rangle u_1 + \langle v_3, u_2 \rangle u_2$

with $\langle v_3, u_1 \rangle u_1 = \left(2/\sqrt{3} + 0 + 1/\sqrt{3} \right) 1/\sqrt{3} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

$= \frac{3}{3} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

and $\langle v_3, u_2 \rangle u_2 = \left(2\times 2 + 0 + 1(-1)\right) \frac{1}{\sqrt{6}} \times \frac{1}{\sqrt{6}} \begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix}$

$$= \frac{1}{2} \begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix}$$

so that $\tilde{u}_3 = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1/2 \\ -1/2 \end{pmatrix}$

and $\| \tilde{u}_3 \| = \sqrt{1 + 1/4 + 1/4} = \frac{\sqrt{6}}{2}$

so that $\boxed{u_3 = \frac{-1}{\sqrt{6}} \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}}$

PROBLEM 2:

(a) $M_V = \begin{pmatrix} 1/2 \\ \vdots \\ 1/2 \end{pmatrix} \left(1/2, \cdots, 1/2 \right) = 1/4 \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$

$\rightarrow$ all columns are the same: rank $M_V = 1$

(b)

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

rank of M_V is 2.

(c) $M_W x = 1/4 \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = 1/4 \begin{pmatrix} 10 \\ 10 \\ 10 \\ 10 \end{pmatrix}$

$M_V x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 0 \\ 0 \end{pmatrix}$ $\neq$

(d)

$M_W M_V =$

$1/4 \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

$= \frac{1}{4} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$

$M_V M_W =$

$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

<u>Tip:</u> You could also use that $(AB)^T = B^T A^T$ noting that M_V and M_W are symmetric matrices.

<u>Intuition:</u> U and V don't overlap except in 0. Projecting last on U will give vectors in U while projecting last on V will give vectors in V. So we expect the order of projection to matter!

(e) $\Pi_V = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

$$\Pi_U \Pi_V = \begin{pmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \Pi_U = \Pi_V \Pi_U$$

They commute because $U \subset V$ this time.

PROBLEM 3:

$$\begin{cases} x' = U^T x & \rightarrow \quad x = U x' \\ y' = U^T y \end{cases}$$

let $x \in \mathbb{R}^n$ and $x' = U^T x$ its coordinates in basis U

let $y = \tilde{L} x$, in the canonical basis, in the "U" basis we have $y' = U^T \tilde{L} x$.

And $x' = U^T x \Rightarrow x = U x'$

(because U is an orthogonal matrix!
$U^{-1} = U^T$
$(U^T)^{-1} = U$)

So that finally $y' = \underbrace{U^T \tilde{L} U}_{} x'$ → transforms in U coordinates.

PROBLEM(X)

(a) as usual

(b) We can use the rank nullity theorem for the linear transformation corresponding to the orthogonal projector on S.

$$\begin{cases} \mathrm{Im}(P_S) = S \\ \ker(P_S) = S^\perp \end{cases}$$

(c) For any $u \in \mathbb{R}^n$ $P_S(u) \in S$ and $u - P_S(u) \in S^\perp$

$$u = P_S(u) + (u - P_S(u))$$
$$= x + y.$$

$\boxtimes$.

DCT_sols

September 16, 2021

1 Compressing images with Discrete Cosine Basis

[1]:
```
%matplotlib inline
import numpy as np
import scipy.fftpack
import scipy.misc
import matplotlib.pyplot as plt
plt.gray()
```

```
<Figure size 432x288 with 0 Axes>
```

[2]:
```
# Two auxiliary functions that we will use. You do not need to read them (but␣
↪make sure to run this cell!)

def dct(n):
    return scipy.fftpack.dct(np.eye(n), norm='ortho')

def plot_vector(v, color='k'):
    plt.plot(v,linestyle='', marker='o',color=color)
```

2 1. Fourier Modes

2.1 1.1. The canonical basis

The vectors of the canonical basis are the columns of the identity matrix in dimension n. We plot their coordinates below for $n = 8$.

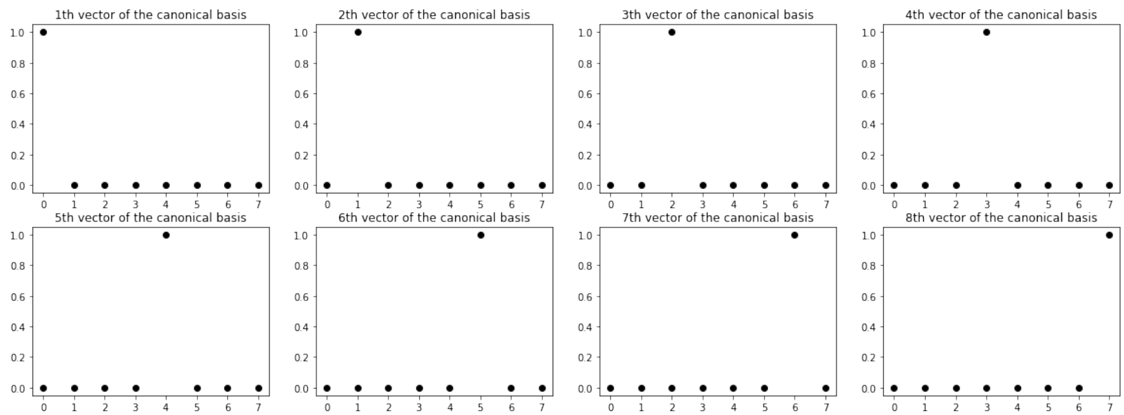
[3]:
```
identity = np.identity(8)
print(identity)

plt.figure(figsize=(20,7))
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.title(f"{i+1}th vector of the canonical basis")
    plot_vector(identity[:,i])

print('\n Nothing new so far...')
```

```
[[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.]]
```

Nothing new so far…



2.2 1.2. Discrete Cosine basis

The discrete Fourier basis is another basis of $\mathbb{R}^n$. The function `dct(n)` outputs a square matrix of dimension n whose columns are the vectors of the discrete cosine basis.

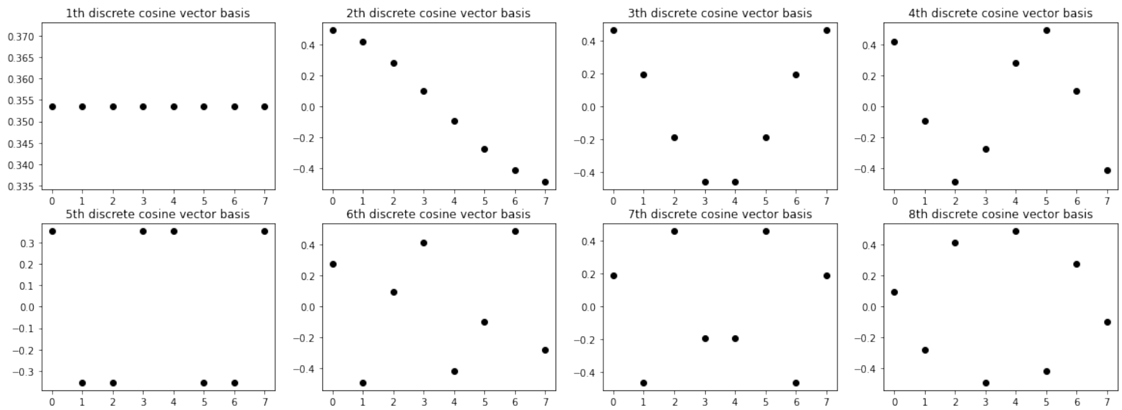
[4]:
```
# Discrete Cosine Transform matrix in dimension n = 8
D8 = dct(8)
print(np.round(D8,3))

plt.figure(figsize=(20,7))

for i in range(8):
    plt.subplot(2,4,i+1)
    plt.title(f"{i+1}th discrete cosine vector basis")
    plot_vector(D8[:,i])
```

```
[[ 0.354  0.49   0.462  0.416  0.354  0.278  0.191  0.098]
 [ 0.354  0.416  0.191 -0.098 -0.354 -0.49  -0.462 -0.278]
 [ 0.354  0.278 -0.191 -0.49  -0.354  0.098  0.462  0.416]
 [ 0.354  0.098 -0.462 -0.278  0.354  0.416 -0.191 -0.49 ]
 [ 0.354 -0.098 -0.462  0.278  0.354 -0.416 -0.191  0.49 ]
 [ 0.354 -0.278 -0.191  0.49  -0.354 -0.098  0.462 -0.416]
```

```
[ 0.354 -0.416  0.191  0.098 -0.354  0.49  -0.462  0.278]
[ 0.354 -0.49   0.462 -0.416  0.354 -0.278  0.191 -0.098]]
```

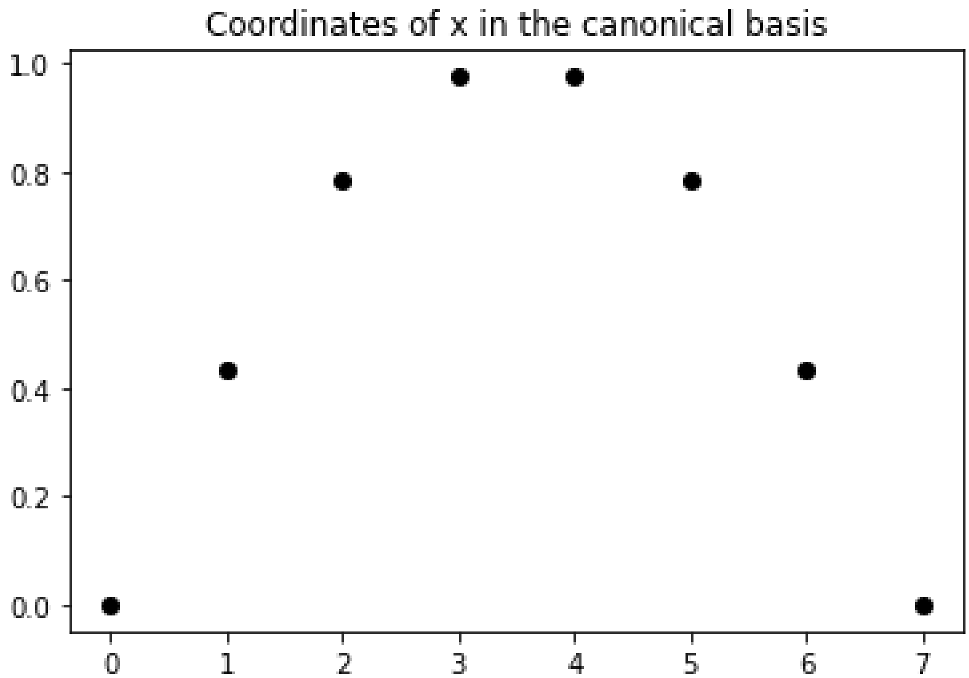


(a) Check numerically (in one line of code) that the columns of `D8` are an orthonormal basis of $\mathbb{R}^8$ (ie verify that the Haar wavelet basis is an orthonormal basis).

[5]:
```python
# Your answer here
np.linalg.norm(D8, axis=0)
np.linalg.norm(D8, axis=1)
np.round(D8 @ D8.T, 3)
```

[5]:
```
array([[ 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.]])
```

[6]:
```python
# Let consider the following vector x
x = np.sin(np.linspace(0,np.pi,8))
plt.title('Coordinates of x in the canonical basis')
plot_vector(x)
```

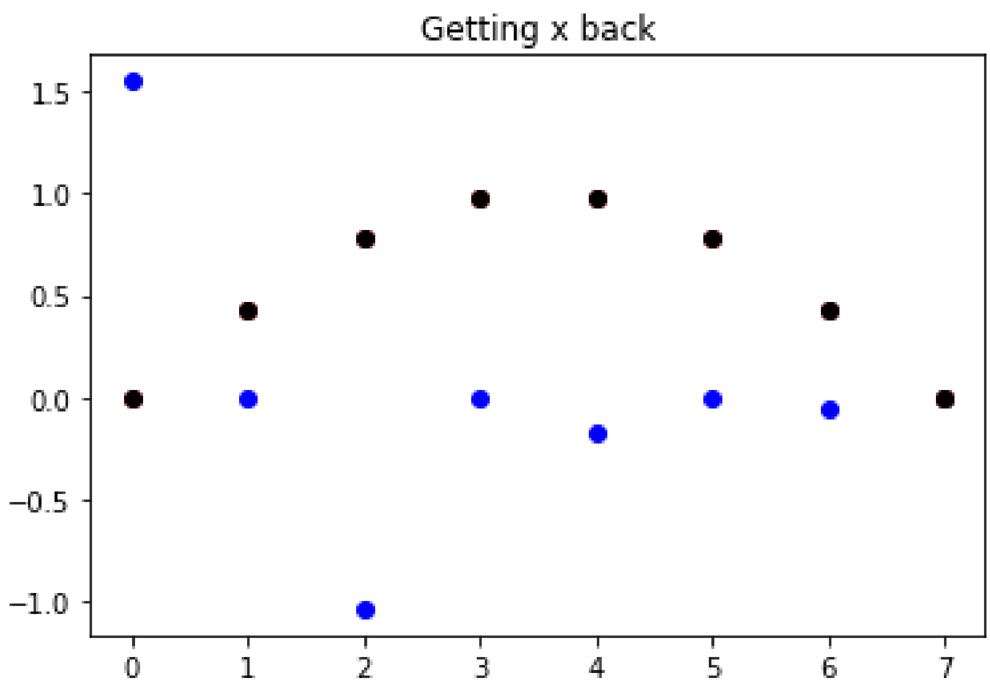


(b) Compute the vector $v \in \mathbb{R}^8$ of DCT coefficients of x. (1 line of code!), and plot them.

How can we obtain back x from v ? (1 line of code!).

[7]:
```
# Write your answer here
plt.title('DCT coefficients of x')
v = D8.T @ x
plot_vector(v, color='b')

plt.title('Getting x back')
xp = D8 @ v
plot_vector(x, color='r')
plot_vector(xp)
```



3 2. Image compression

In this section, we will use DCT modes to compress images.

[8]:
```
image = scipy.misc.face(gray=True)
h,w = image.shape
print(f'Height: {h}, Width: {w}')

plt.imshow(image)
```

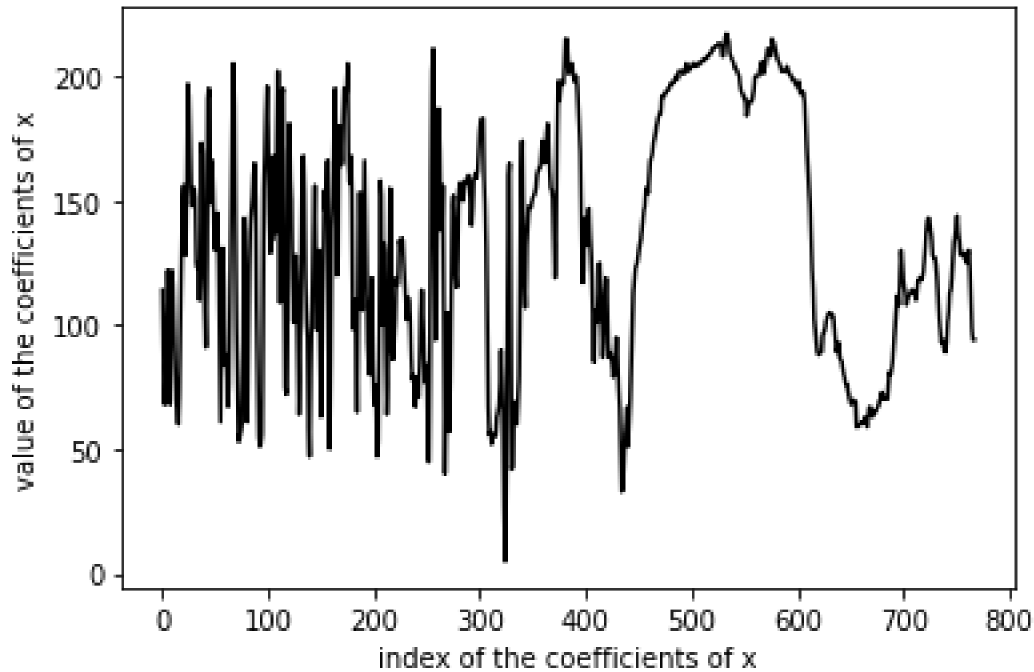
```
Height: 768, Width: 1024
```

[8]: `<matplotlib.image.AxesImage at 0x7fb590d7af10>`

4 Image compression

We will see each column of pixels as a vector in $\mathbb{R}^{768}$, and compute their coordinates in the DCT basis of $\mathbb{R}^{768}$. Plot the entries of `x`, the first column of our image.

[9]:
```
# Your answer here
image = scipy.misc.face(gray=True)
x = image[:,0]
plt.plot(x, color='black')
plt.xlabel('index of the coefficients of x')
plt.ylabel('value of the coefficients of x')
plt.show()
```

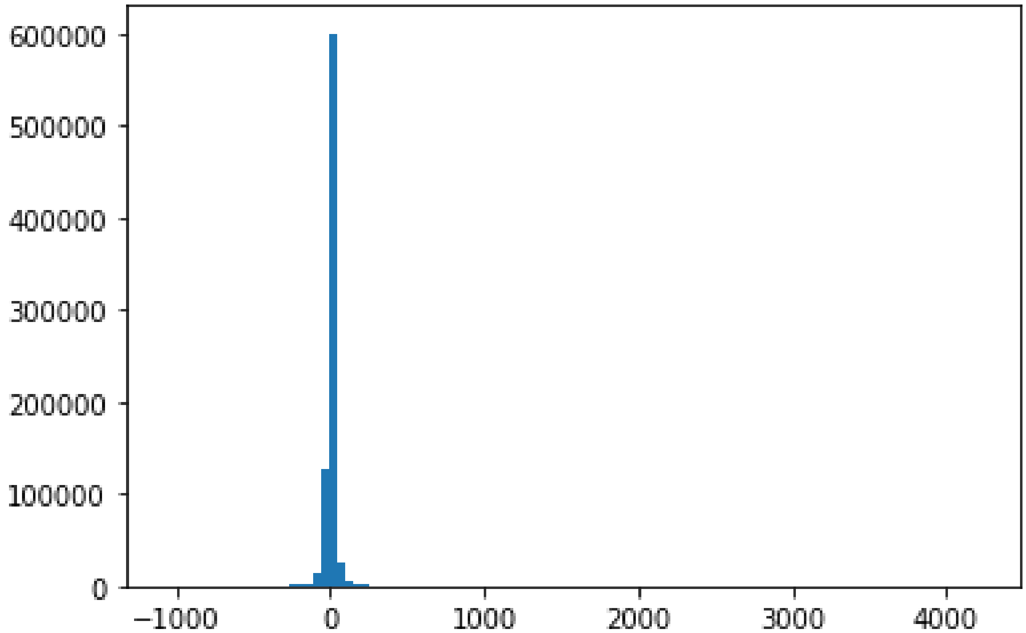


(d) Compute the 768 x 1024 matrix `dct_coeffs` whose columns are the dct coefficients of the columns of `image`. Plot an histogram of there intensities using `plt.hist`.

[10]:
```
# Your answer here
D = dct(768)
dct_coeffs = D.T @ image
plt.hist(dct_coeffs.flatten(), 100)
```

[10]:
```
(array([5.00000e+00, 2.60000e+01, 2.20000e+01, 3.20000e+01, 3.00000e+01,
        7.80000e+01, 7.50000e+01, 8.70000e+01, 1.11000e+02, 1.70000e+02,
        2.20000e+02, 3.88000e+02, 4.29000e+02, 5.00000e+02, 7.91000e+02,
        9.60000e+02, 1.59400e+03, 3.54000e+03, 1.28690e+04, 1.25898e+05,
        5.99967e+05, 2.64150e+04, 5.10800e+03, 2.14700e+03, 1.08700e+03,
        7.32000e+02, 5.62000e+02, 4.90000e+02, 3.33000e+02, 1.83000e+02,
        1.09000e+02, 9.20000e+01, 6.80000e+01, 6.50000e+01, 5.00000e+01,
        3.60000e+01, 4.70000e+01, 2.70000e+01, 1.50000e+01, 1.00000e+01,
        1.10000e+01, 3.00000e+00, 1.00000e+00, 5.00000e+00, 8.00000e+00,
        7.00000e+00, 5.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
        0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
        0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
        0.00000e+00, 2.00000e+00, 3.20000e+01, 4.80000e+01, 2.70000e+01,
        1.30000e+01, 1.20000e+01, 3.90000e+01, 5.70000e+01, 3.70000e+01,
        5.00000e+01, 4.40000e+01, 5.40000e+01, 1.30000e+01, 5.00000e+00,
        6.00000e+00, 1.10000e+01, 1.40000e+01, 1.00000e+01, 3.40000e+01,
        1.90000e+01, 1.70000e+01, 1.20000e+01, 1.00000e+01, 2.30000e+01,
```

```
       2.20000e+01, 2.70000e+01, 4.80000e+01, 4.00000e+01, 5.10000e+01,
       4.10000e+01, 4.40000e+01, 3.70000e+01, 2.90000e+01, 2.20000e+01,
       2.80000e+01, 2.50000e+01, 1.00000e+01, 5.00000e+00, 6.00000e+00]),
array([-1064.43123878, -1011.70999962,  -958.98876045,  -906.26752129,
        -853.54628213,  -800.82504297,  -748.1038038 ,  -695.38256464,
        -642.66132548,  -589.94008632,  -537.21884715,  -484.49760799,
        -431.77636883,  -379.05512967,  -326.33389051,  -273.61265134,
        -220.89141218,  -168.17017302,  -115.44893386,   -62.72769469,
         -10.00645553,    42.71478363,    95.43602279,   148.15726196,
         200.87850112,   253.59974028,   306.32097944,   359.04221861,
         411.76345777,   464.48469693,   517.20593609,   569.92717526,
         622.64841442,   675.36965358,   728.09089274,   780.81213191,
         833.53337107,   886.25461023,   938.97584939,   991.69708856,
        1044.41832772,  1097.13956688,  1149.86080604,  1202.5820452 ,
        1255.30328437,  1308.02452353,  1360.74576269,  1413.46700185,
        1466.18824102,  1518.90948018,  1571.63071934,  1624.3519585 ,
        1677.07319767,  1729.79443683,  1782.51567599,  1835.23691515,
        1887.95815432,  1940.67939348,  1993.40063264,  2046.1218718 ,
        2098.84311097,  2151.56435013,  2204.28558929,  2257.00682845,
        2309.72806762,  2362.44930678,  2415.17054594,  2467.8917851 ,
        2520.61302427,  2573.33426343,  2626.05550259,  2678.77674175,
        2731.49798091,  2784.21922008,  2836.94045924,  2889.6616984 ,
        2942.38293756,  2995.10417673,  3047.82541589,  3100.54665505,
        3153.26789421,  3205.98913338,  3258.71037254,  3311.4316117 ,
        3364.15285086,  3416.87409003,  3469.59532919,  3522.31656835,
        3575.03780751,  3627.75904668,  3680.48028584,  3733.201525  ,
        3785.92276416,  3838.64400333,  3891.36524249,  3944.08648165,
        3996.80772081,  4049.52895997,  4102.25019914,  4154.9714383 ,
        4207.69267746]),
<BarContainer object of 100 artists>)
```



Since a large fraction of the dct coefficients seems to be negligible, we see that the vector `x` can be well approximated by a linear combination of a small number of discrete cosines vectors.

Hence, we can 'compress' the image by only storing a few dct coefficients of largest magnitude.

Let's say that we want to reduce the size by 98%: Store only the top 2% largest (in absolute value) coefficients of `wavelet_coeffs`.

(e) Compute a matrix `thres_coeffs` who is the matrix `dct_coeffs` where about 97% smallest entries have been put to 0.

[53]:
```
# Your answer here
idx = int(dct_coeffs.flatten().shape[0] * 0.02)
thres = np.sort(np.abs(dct_coeffs).flatten())[::-1][idx]
thres
# idx
thres_coeffs = dct_coeffs
thres_coeffs = thres_coeffs * (dct_coeffs < thres)
```

[53]:
```
<matplotlib.image.AxesImage at 0x7fb57c3ef4d0>
```

(f) Compute and plot the `compressed_image` corresponding to `thres_coeffs`.

[54]:
```
# Your answer here
xp = D @ thres_coeffs


plt.imshow(xp)
# plt.hist(thres_coeffs.flatten(), 100)
```

[54]:
```
<matplotlib.image.AxesImage at 0x7fb57cba3750>
```


0
100
200
300
400
500
600
700
0
200
400
600
800
1000