Compressing images with Discrete Cosine Basis

In [ ]:

```
%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>
```

In [ ]:

```
# Two auxiliary functions that we will use. You do not need to read them (but ma

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)
```

5.3.1 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 $n = 8$.

In [ ]:

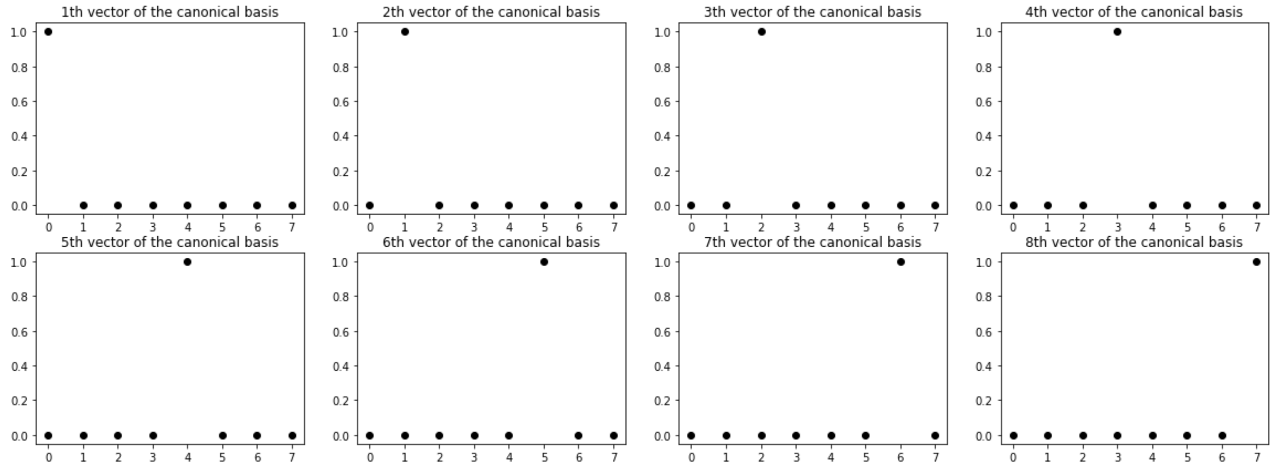
```
identity = np.identity(8)
print(identity)

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...
```



5.3.2 Discrete Cosine basis

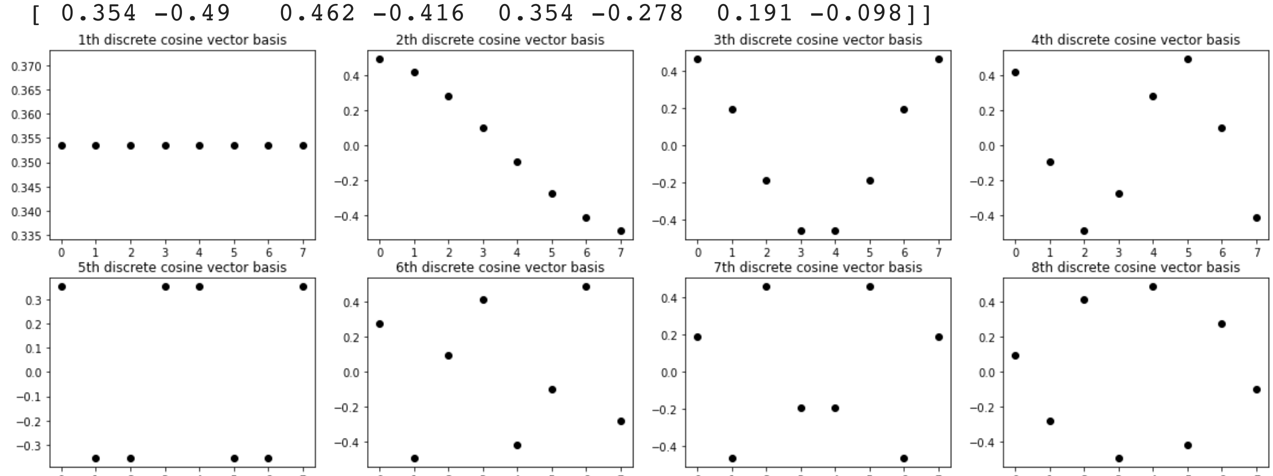
The discrete Fourier basis is another basis of $\mathbb{R}^n$. The function `dct(n)` outputs a square matrix of dimension n whose columns are the vectors of the discrete cosine basis.

In [ ]:
```
# 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]]
```

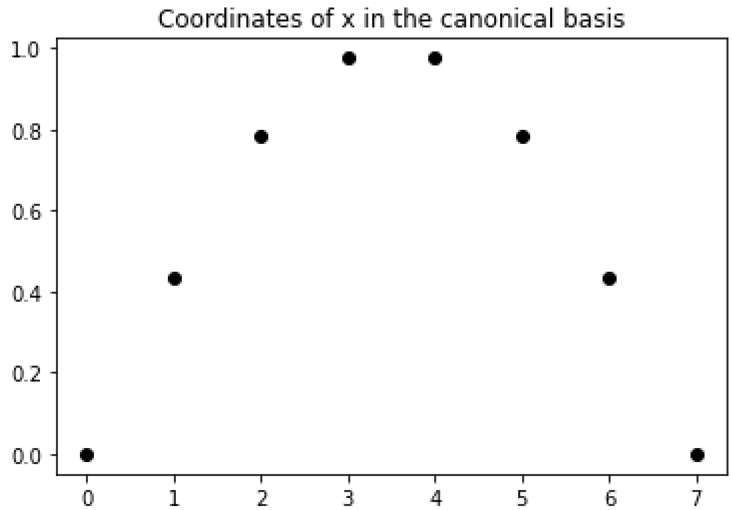


5.3 (a) Check numerically (in one line of code) that the columns of `D8` are an orthonormal basis of $\mathbb{R}^8$ (ie verify that the Haar wavelet basis is an orthonormal basis).

In [ ]:
```
# Your answer here
print((D8 @ np.transpose(D8)).round(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.  0.]
 [ 0.  0.  0. -0.  0.  0.  0.  1.]]
```

In [ ]:
```
# 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)
```



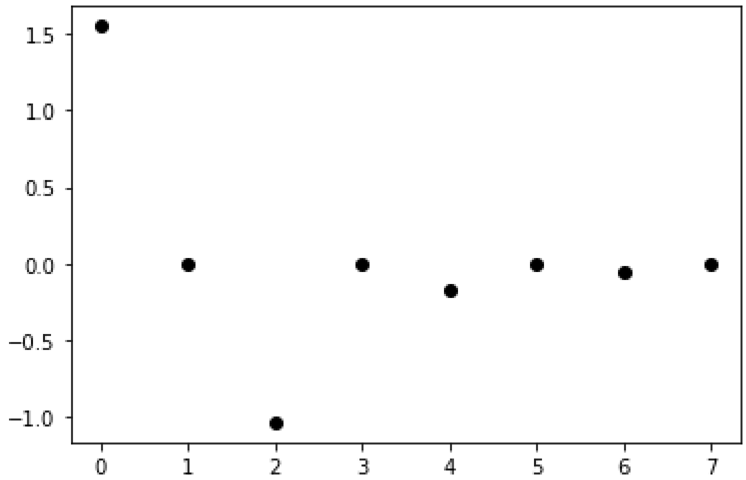
5.3 (b) Compute the vector $v \in \mathbb{R}^8$ of DCT coefficients of x. (1 line of code!), and plot them.

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

In [ ]:
```
# Write your answer here
plot_vector(np.linalg.inv(D8) @ x)

(D8) @ ((np.linalg.inv(D8) @ x)).round(3)
```

Out[ ]:
```
array([-4.53619217e-05,  4.33931812e-01,  7.81584745e-01,  9.75145613e-01,
        9.75145613e-01,  7.81584745e-01,  4.33931812e-01, -4.53619217e-05])
```



5.3.3 Image compression

In this section, we will use DCT modes to compress images. Let's use one of the template images of python.

In [ ]:

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

plt.imshow(image)
```

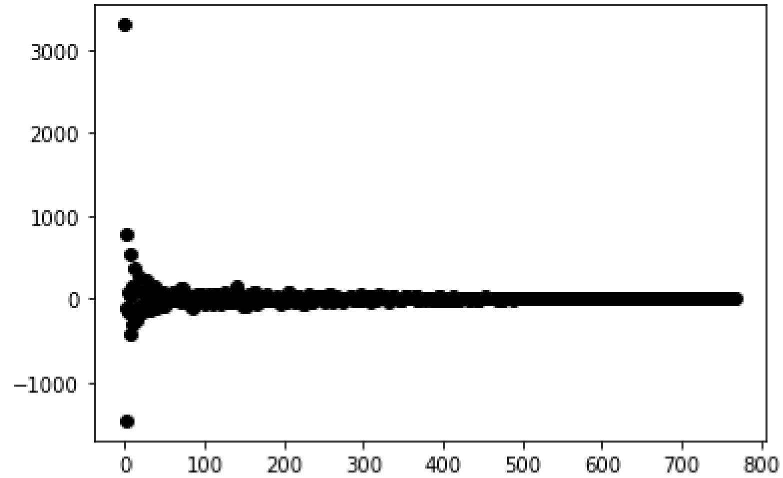
```
Height: 768, Width: 1024
```

Out[ ]: `<matplotlib.image.AxesImage at 0x7f86984050d0>`

5.3 (c) 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.

In [ ]:

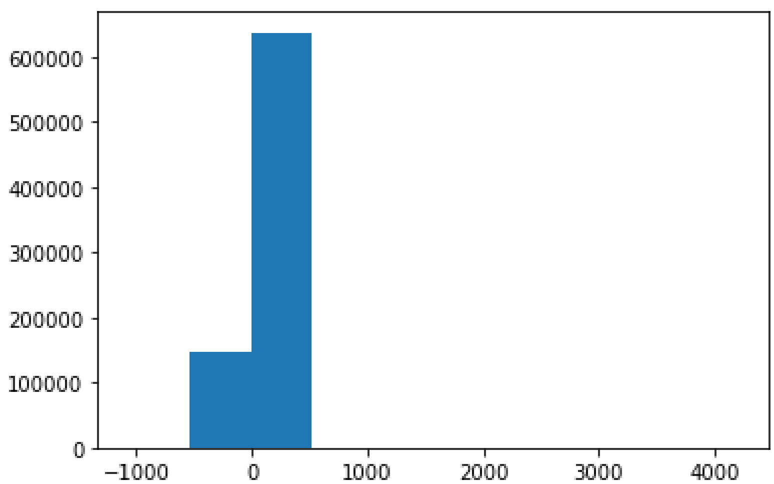
```
# Your answer here
x = image[:,1] @ np.linalg.inv(dct(768))
plot_vector(x)
```



5.3 (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`.

In [ ]:
```
# Your answer here
coeffs = (np.linalg.inv(dct(768)) @ image).flatten()
plt.hist(coeffs)
```

Out[ ]:
```
(array([6.36000e+02, 1.47189e+05, 6.37024e+05, 5.19000e+02, 4.00000e+01,
        0.00000e+00, 2.67000e+02, 2.41000e+02, 2.69000e+02, 2.47000e+02]),
 array([-1064.43123878,  -537.21884715,   -10.00645553,   517.20593609,
         1044.41832772,  1571.63071934,  2098.84311097,  2626.05550259,
         3153.26789421,  3680.48028584,  4207.69267746]),
 <BarContainer object of 10 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`.

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

In [ ]:
```
# Your answer here
```

```
coeffs = np.abs(coeffs)
coeffs.sort()
coeffs = coeffs[::-1]
threshhold = coeffs[23593]

# dct768 = np.linalg.inv(dct(768)) @ image
dct768 = dct(768) @ image
dct768[np.abs(dct768)<threshhold] = 0
dct768 = np.linalg.inv(dct(768)) @ dct768

threes_coeffs = dct768
threes_coeffs
```

Out[ ]:

```
array([[ 64.99578887,  79.44342592,  80.71002817, ...,  91.2056512 ,
         98.09297156, 104.13929048],
       [ 91.06507828, 111.07107303, 110.50154926, ..., 128.7875538 ,
        138.20307674, 146.69260451],
       [ 88.81271141, 107.61613455, 100.96029168, ..., 128.19044403,
        136.66707391, 144.97402188],
       ...,
       [ 56.70854004,  52.60615976,  56.72224744, ...,  56.44237312,
         52.21441327,  53.98436896],
       [ 40.01834057,  37.61525326,  41.25723988, ...,  38.70584614,
         35.58375021,  36.78324651],
       [ 20.74578708,  19.66026933,  21.82342459, ...,  19.68618258,
         18.02965993,  18.6353314 ]])
```

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

In [ ]:

```
# Your answer here
plt.imshow(threes_coeffs)
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7f86b951df40>
```

In [ ]: