Compressing images with Discrete Cosine Basis

In [103…

```
%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 [104…

```
# 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 [105…

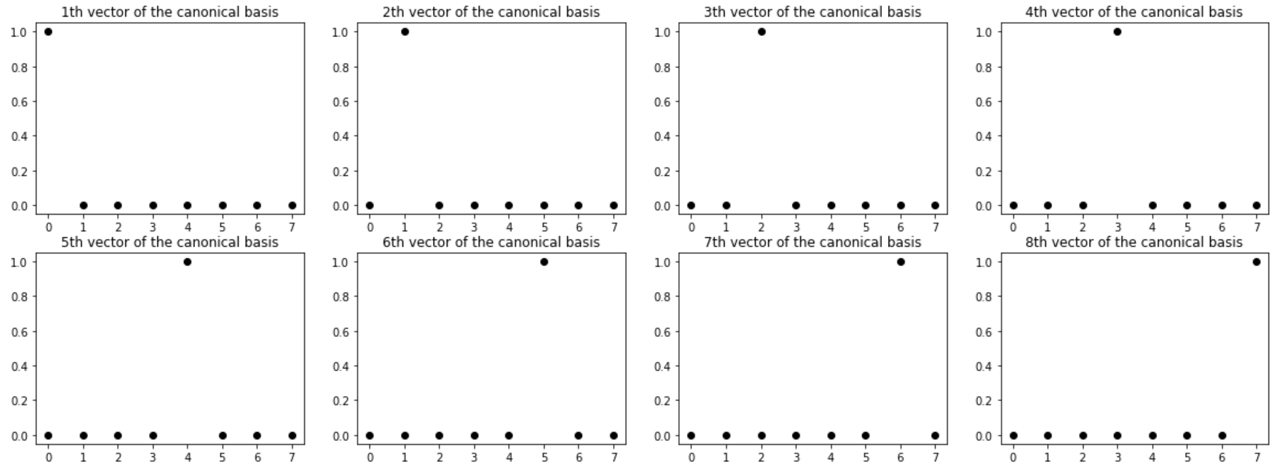
```
identity = np.identity(8)
print(identity)

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.

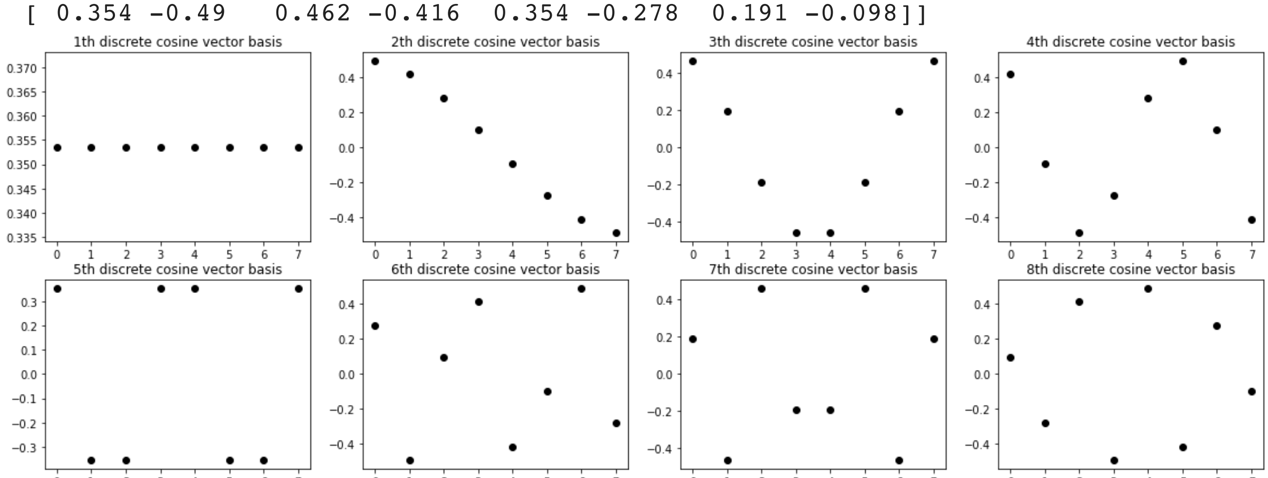
In [106…

```
# 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]]
```

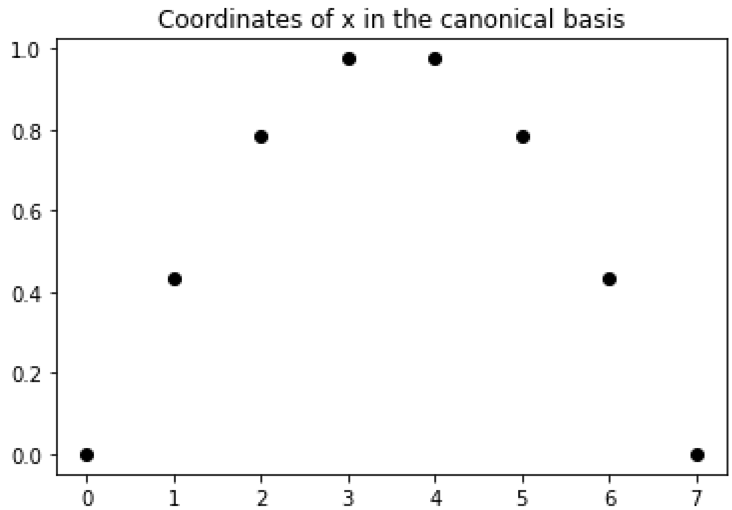


5.3 (a) Check numerically (in one line of code) that the columns of `D8` are an orthonormal basis of $\mathbb{R}^8$ (ie verify that the Haar wavelet basis is an orthonormal basis).

In [107…
```
[True]*8 == [round(np.linalg.norm(D8[:,i]), 5) == 1 for i in range(8)]
# If output is True, then the columns of D8 are an orthonormal basis of R^8
```

Out[107…
```
True
```

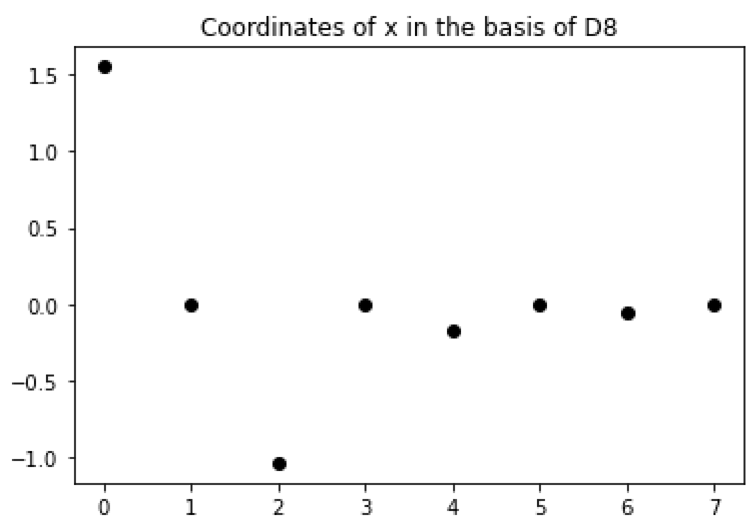
In [108…
```
# Let consider the following vector x
x = np.sin(np.linspace(0,np.pi,8))
plt.title('Coordinates of x in the canonical basis')
plot_vector(x)
```



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 [109…
```
v = (np.linalg.inv(D8))@x
plt.title('Coordinates of x in the basis of D8')
plot_vector(v)
x = (D8)@v
```



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 [110…

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

plt.imshow(image)
```

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

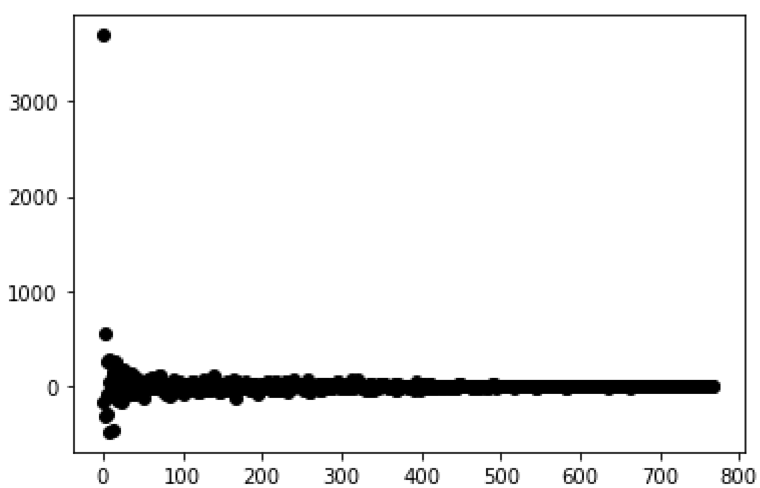
Out[110…

```
<matplotlib.image.AxesImage at 0x7ff92384af70>
```

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 [111…

```
D768 = dct(768)
x = (np.linalg.inv(D768))@(image[:,0])
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 [118…

```
dct_coeffs = (np.linalg.inv(D768))@image
plt.hist(dct_coeffs)
```

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 [122…

```
entries = dct_coeffs.tolist()
entries = [abs(item) for sublist in entries for item in sublist]
entries.sort()
cutoff = entries[int(round(0.97*len(entries), 0))]
print(cutoff)

thres_coeffs = dct_coeffs
for i in range(1024):
    for j in range(768):
        if abs(thres_coeffs[j,i]) <= cutoff:
            thres_coeffs[j,i] = 0
```

```
96.87791323323603
```

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

In [123…

```
compressed_image = D768@dct_coeffs
plt.imshow(compressed_image)
```

Out[123…

```
<matplotlib.image.AxesImage at 0x7ff9646cc9a0>
```