Lab session 5

Discrete cosine transform
Introduction to JPEG

report by Grebennikova Anna & Kseniia Ovchinnikova

Université Grenoble Alpes, UGA

Exercise 1. *Discrete cosine transform*

Let f be a discrete signal defined on $\{0, \cdots, L-1\}$, consider the signal $\tilde{f}$ built by symmetrizing f with respect to $-\frac{1}{2}$, namely

$$\tilde{f}_n = \begin{cases} f_n & \text{for } 0 \leq n < L, \\ f_{-n-1} & \text{for } -L \leq n \leq -1. \end{cases}$$

We then assume $\tilde{f}$ is periodized with period $2L$.

Question 1.1

We aim to prove that f_n admits the following decomposition:

$$f_n = \sum_{k=0}^{L-1} a_k \cos\left(\frac{k\pi}{L}\left(n + \frac{1}{2}\right)\right)$$

To establish this, let us calculate f_{-n-1}:

$$f_{-n-1} = \sum_{k=0}^{L-1} a_k \cos\left(\frac{k\pi}{L}\left(-n-1+\frac{1}{2}\right)\right) = \sum_{k=0}^{L-1} a_k \cos\left(\frac{k\pi}{L}\left(-n-\frac{1}{2}\right)\right) = \sum_{k=0}^{L-1} a_k \cos\left(\frac{k\pi}{L}\left(n+\frac{1}{2}\right)\right) = f_n$$

Therefore, we conclude:

$$\Rightarrow f_n = f_{-n-1}$$

This result demonstrates that f_n possesses a symmetry property, namely $f_n = f_{-n-1}$. Consequently, the function f_n is symmetric about $-\frac{1}{2}$, and the given decomposition holds true. The decomposition effectively captures the periodic and symmetric nature of the discrete signal f_n.

To determine the coefficients a_k, we use the following approach:

$$f_n = \frac{\tilde{f}_n + \tilde{f}_{-n-1}}{2}$$

We know that

$$\tilde{f}_n = \frac{1}{2L} \sum_{k=0}^{2L-1} \hat{\tilde{f}}_k e^{\frac{2i\pi kn}{2L}}$$

Substituting this expression for $\tilde{f}_n$ and $\tilde{f}_{-n-1}$ into the formula for f_n, we obtain:

$$f_n = \frac{1}{4L}\left(\sum_{k=0}^{2L-1} \hat{\tilde{f}}_k e^{\frac{2i\pi kn}{2L}} + \sum_{k=0}^{2L-1} \hat{\tilde{f}}_k e^{\frac{2i\pi k(-n-1)}{2L}}\right) = \frac{1}{4L}\sum_{k=0}^{2L-1} \hat{\tilde{f}}_k \left(e^{\frac{2i\pi kn}{2L}} + e^{-\frac{2i\pi kn}{2L}} e^{-\frac{2i\pi k}{2L}}\right) =$$

$$= \frac{1}{4L}\sum_{k=0}^{2L-1} \hat{\tilde{f}}_k e^{-\frac{i\pi k}{2L}} \left(e^{\frac{2i\pi k}{2L}\left(n+\frac{1}{2}\right)} + e^{-\frac{2i\pi k}{2L}\left(n+\frac{1}{2}\right)}\right) = \frac{1}{4L}\sum_{k=0}^{2L-1} \hat{\tilde{f}}_k e^{-\frac{i\pi k}{2L}} \cdot 2\cos\left(\frac{\pi k}{L}\left(n+\frac{1}{2}\right)\right)$$

$$= \frac{1}{2L}\left(\sum_{k=0}^{L-1} \hat{\tilde{f}}_k e^{-\frac{i\pi k}{2L}} \cos\left(\frac{\pi k}{L}\left(n+\frac{1}{2}\right)\right) + \sum_{k=1}^{L-1} \hat{\tilde{f}}_{2L-k} e^{-\frac{i\pi(2L-k)}{2L}} \cos\left(\frac{\pi(2L-k)}{L}\left(n+\frac{1}{2}\right)\right)\right)$$

Next, we simplify the expression for the second sum:

$$e^{\frac{-i\pi(2L-k)}{2L}} \cos\left(\frac{\pi(2L-k)\left(n+\frac{1}{2}\right)}{L}\right) = e^{-i\pi} e^{\frac{i\pi k}{2L}} \cos\left(2\pi n + \pi - \frac{\pi k}{L}\left(n+\frac{1}{2}\right)\right) =$$

$$= -e^{\frac{i\pi k}{2L}} \cdot \left(-\cos\left(\frac{\pi k}{L}\left(n+\frac{1}{2}\right)\right)\right) = e^{\frac{i\pi k}{2L}} \cos\left(\frac{\pi k}{L}\left(n+\frac{1}{2}\right)\right)$$

Using this result, we obtain:

$$f_n = \frac{1}{2L}\left(\widehat{\tilde{f}}_0 + \sum_{k=0}^{L-1} \widehat{\tilde{f}}_k e^{-\frac{i\pi k}{2L}} \cos\left(\frac{\pi k}{L}\left(n+\frac{1}{2}\right)\right) + \sum_{k=1}^{L-1} \widehat{\tilde{f}}_{2L-k} e^{\frac{i\pi k}{2L}} \cos\left(\frac{\pi k}{L}\left(n+\frac{1}{2}\right)\right) + \widehat{\tilde{f}}_{2L}\right)$$

$$= \frac{1}{2L}\sum_{k=0}^{L-1}\left(\widehat{\tilde{f}}_k e^{-i\frac{\pi k}{2L}} + \widehat{\tilde{f}}_{2L-k} e^{i\frac{\pi k}{2L}}\right)\cos\left(\frac{\pi k}{L}\left(n+\frac{1}{2}\right)\right)$$

Finally, the coefficients a_k are given by:

$$a_k = \frac{\widehat{\tilde{f}}_k e^{-i\frac{\pi k}{2L}} + \widehat{\tilde{f}}_{2L-k} e^{i\frac{\pi k}{2L}}}{2L}$$

Question 1.2

To demonstrate that this basis is orthogonal, we compute the inner product $_k, e_j\rangle$ for two basis vectors. We analyze two cases: when $k = j$ (self-product) and when $k \neq j$ (cross-product).

The inner product is defined as:

$$\langle e_k, e_j\rangle = \sum_{n=0}^{L-1} \lambda_k \sqrt{\frac{2}{L}} \cos\left(\frac{k\pi}{L}\left(n+\frac{1}{2}\right)\right) \lambda_j \sqrt{\frac{2}{L}} \cos\left(\frac{j\pi}{L}\left(n+\frac{1}{2}\right)\right) =$$

$$= \frac{\lambda_k\lambda_j 2}{L}\sum_{n=0}^{L-1}\frac{\cos\left(\frac{\pi}{L}(k+j)\left(n+\frac{1}{2}\right)\right) + \cos\left(\frac{\pi}{L}(k-j)\left(n+\frac{1}{2}\right)\right)}{2}$$

$$= \frac{\lambda_k\lambda_j}{L}\left(\sum_{n=0}^{L-1}\cos\left(\frac{\pi}{L}(k+j)\left(n+\frac{1}{2}\right)\right) + \sum_{n=0}^{L-1}\cos\left(\frac{\pi}{L}(k-j)\left(n+\frac{1}{2}\right)\right)\right)$$

If $k \neq j$, summing both cosine terms will yield 0 because they form a symmetric function.

If $k = j \neq 0$:

$$\langle e_k, e_k\rangle = \frac{\lambda_k^2}{L}\left(\sum_{n=0}^{L-1}\cos\left(\frac{2k\pi}{L}\left(n+\frac{1}{2}\right)\right) + L\right) = \frac{1}{L}(0+L) = \frac{1}{L}\cdot L = 1$$

If $k = j = 0$:

$$\langle e_0, e_0\rangle = \frac{\lambda_k^2}{L}(L+L) = \frac{1}{2L}\cdot 2L = 1$$

Question 1.3

Decomposition of f_n in terms of the basis:

$$f_n = \sum_{k=0}^{L-1} a_k e_k(n)$$

Since $\{e_k\}$ is orthonormal, we can calculate the coefficients a_k as:

$$a_k = \langle f, e_k\rangle = \sum_{n=0}^{L-1} f_n e_k(n)$$

Substituting the definition of $e_k(n)$:

$$a_k = \lambda_k \sqrt{\frac{2}{L}} \sum_{n=0}^{L-1} f_n \cos\left(\frac{k\pi}{L}\left(n+\frac{1}{2}\right)\right)$$

Question 1.4

The program to decompose a signal f in the discrete cosine transform:

```python
def dct_transform(f):
    L = len(f)
    a_k = np.zeros(L)  # Array to store DCT coefficients
    for k in range(L):
        # Definition of _k
        if k == 0:
            lambda_k = 1 / np.sqrt(2)
        else:
            lambda_k = 1
        # Compute the coefficient a_k
        for n in range(L):
            a_k[k] += lambda_k * np.sqrt(2 / L) * f[n] * np.cos((np.pi * k / L) * (n + 0.5))
    return a_k
```

Question 1.5

Suppose we have two vectors:

- $\mathbf{u} \in \mathbb{R}^m$, written as $\mathbf{u} = [u_1, u_2, \ldots, u_m]$,
- $\mathbf{v} \in \mathbb{R}^n$, written as $\mathbf{v} = [v_1, v_2, \ldots, v_n]$.

Then their tensor product $\mathbf{u} \otimes \mathbf{v}$ is a new vector of size $m \times n$, whose elements are defined as:

$$\mathbf{u} \otimes \mathbf{v} = \begin{bmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_n \\ u_2v_1 & u_2v_2 & \cdots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \cdots & u_mv_n \end{bmatrix}.$$

Thus, using a tensor product of 1D basis we get an orthonormal basis for images of size $L \times L$:

$$e_{k_1,k_2}(n_1, n_2) = \lambda_{k_1}\lambda_{k_2}\frac{2}{L}\cos\left(\frac{k_1\pi}{L}\left(n_1 + \frac{1}{2}\right)\right)\cos\left(\frac{k_2\pi}{L}\left(n_2 + \frac{1}{2}\right)\right),$$

where:

- n_1, n_2 are the coordinates of a pixel in the image,
- k_1, k_2 are the indices of the basis functions.

The image function $f(n_1, n_2)$ can be represented as:

$$f(n_1, n_2) = \sum_{k_1=0}^{L-1}\sum_{k_2=0}^{L-1} a_{k_1,k_2}e_{k_1,k_2}(n_1, n_2),$$

where the coefficients a_{k_1,k_2} could be computed as:

$$a_{k_1,k_2} = \sum_{n_1=0}^{L-1}\sum_{n_2=0}^{L-1} f(n_1, n_2)e_{k_1,k_2}(n_1, n_2).$$

Exercise 2. *Introduction to JPEG*

Question 2.1

We write a program to read an image named `cameraman.png` and compute its DCT transform. The orthonormal basis at each point (n_1, n_2) is computed using an auxiliary function.

In our case, the original matrix contains 256×256 elements. When we split it into blocks of 8×8, we obtain:

$$\left(\frac{256}{8}\right)^2 = 32 \times 32 = 1024$$

matrices. These blocks are stored as a 3D array with dimensions $(1024, 8, 8)$.

```python
import numpy as np
import matplotlib.pyplot as plt

def DCT(f):
  ak = np.zeros_like(f)
  L = len(f)

  for k1 in range(L):
    lambda_k1 = 1 / np.sqrt(2) if k1 == 0 else 1
    for k2 in range(L):
      lambda_k2 = 1 / np.sqrt(2) if k1 == 0 else 1
      for n1 in range(L):
        for n2 in range(L):
          ak[k1,k2] += lambda_k1 * lambda_k2 * 2 / L * f[n1,n2] * np.cos((np.pi * k1 / L) * (n1 + 0.5)) * np.cos((np.pi * k2 / L) * (n2 + 0.5))
  return ak

# Processing of image
image = plt.imread('cameraman.png')  # Load the image (convert to grayscale if necessary)
image_array = np.array(image)  # Convert the image to a NumPy array

# Block size
block_size = 8

# Create a list to store all the blocks
blocks = []

# Split the image into 8x8 blocks
for i in range(0, 256, block_size):
    for j in range(0, 256, block_size):
        block = image_array[i:i + block_size, j:j + block_size]
        blocks.append(block)

blocks_array = np.array(blocks)

# Performing the DCT on each block
blocks_array_DCT = np.zeros(np.shape(blocks_array))
for i in range(blocks_array.shape[0]):
    blocks_array_DCT[i] = DCT(blocks_array[i])
```

Question 2.2

The coefficients a_k for each block were computed using the formula derived in **Question 1.5**.

As a result of the program described in **Question 2.1**, we obtain the following coefficients for the first 8×8 matrix out of the 1024 matrices.

$$\begin{bmatrix} 4.9211 & -0.0055 & 0.0018 & 0.0013 & 0.0066 & 0.0029 & 0.0071 & -0.0014 \\ 0.0212 & -0.0028 & 0.0021 & 0.0028 & -0.0077 & 0.0008 & -0.0090 & 0.0157 \\ 0.0113 & 0.0020 & 0.0132 & -0.0057 & -0.0093 & -0.0062 & -0.0019 & -0.0026 \\ -0.0277 & 0.0003 & -0.0124 & -0.0018 & 0.0036 & -0.0036 & 0.0021 & -0.0011 \\ 0.0187 & -0.0017 & 0.0013 & 0.0024 & 0.0034 & 0.0011 & -0.0085 & 0.0021 \\ -0.0115 & 0.0036 & 0.0044 & -0.0056 & -0.0100 & 0.0002 & 0.0015 & 0.0056 \\ 0.0111 & -0.0029 & 0.0040 & -0.0054 & -0.0084 & -0.0213 & -0.0083 & -0.0030 \\ -0.0159 & 0.0039 & -0.0050 & -0.0080 & -0.0050 & -0.0016 & -0.0149 & 0.0123 \end{bmatrix}$$

Question 2.3

We compute the quantization matrix $Q(i,j) = 1 + q(1 + i + j)$ with the use of auxiliary function:

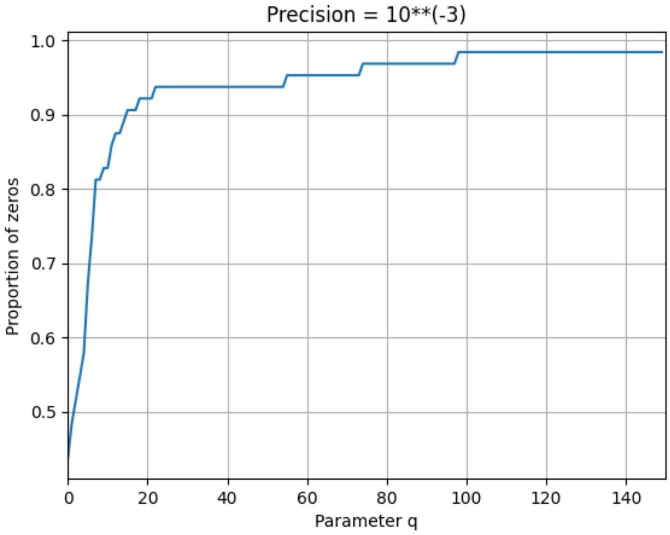
```python
def quan_matrix(q,block_size):
  Q = np.zeros((block_size,block_size))
  for i in range(block_size):
    for j in range(block_size):
      Q[i,j] = 1 + q * (1 + i + j)
  return Q
```

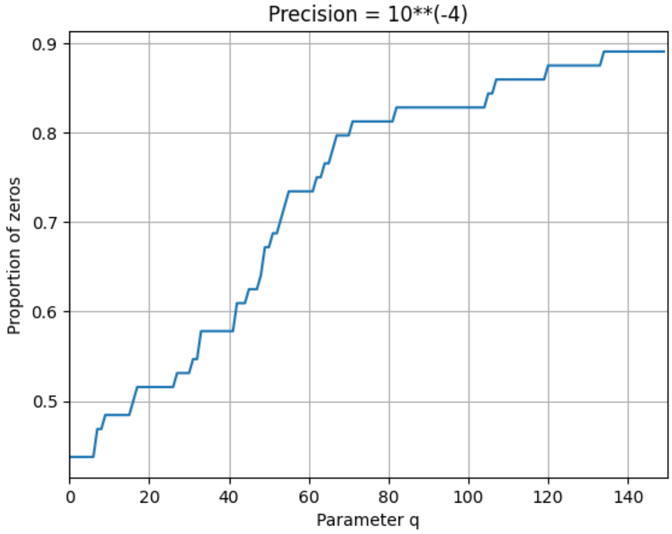
Question 2.4

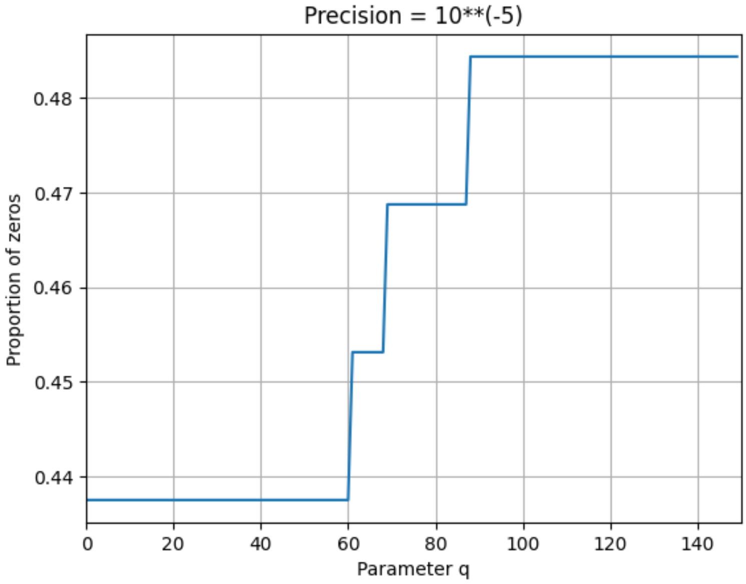
We plot the proportion of zero coefficients as a function of q. The most important parameter here is precision, because the matrix does not contain exact zeros but rather very small values. If a value is smaller than the precision, we assume it is equal to zero.

There are three graphs, corresponding to precision values of 10^{-3}, 10^{-4}, and 10^{-5}, respectively.

```python
# Define the maximum range of the quantization parameter q
ax_len = 150
proportion_of_zeros = np.zeros(ax_len)

# Define the precision threshold below which values are considered zero
precision = 10**(-4)

blocks_array_DCT_quan = np.zeros(np.shape(blocks_array))

# Loop through different quantization levels (q values)
for q in range(0, ax_len):
    # Generate the quantization matrix for the current q value
    Q = quan_matrix(q, block_size)

    for i in range(blocks_array_DCT.shape[0]):
        # Quantize the DCT coefficients by dividing by the quantization matrix Q
        blocks_array_DCT_quan[i] = blocks_array_DCT[i] / Q

    # Calculate the proportion of coefficients that are effectively zero (less than
    ↪ precision)
    proportion_of_zeros[q] = np.sum(blocks_array_DCT_quan[i] < precision) / (block_size**2)
```







We observe that with the increase of q, the number of zero elements rises significantly. For precision 10^{-3}, the proportion of zeros grows rapidly, and at a compression ratio of approximately 20, the number of zeros exceeds 90%.

For precision 10^{-4}, the graph grows more gradually, reaching about 90% zeros at a compression ratio of 120. However, for precision 10^{-5}, the quantization is insufficient to make the coefficients smaller than this threshold.

Question 2.5

In the DCT matrix, low-frequency coefficients, located in the top-left corner, typically have larger magnitudes, while high-frequency coefficients, in the bottom-right, are much smaller. During quantization, a quantization matrix is applied to reduce the precision of the coefficients, where higher-frequency coefficients are more aggressively quantized due to their smaller magnitudes. This often results in many high-frequency coefficients being rounded to zero. The zigzag order organizes the coefficients such that the most significant, low-frequency coefficients appear first, followed by the high-frequency coefficients, which are more likely to be zero.