Assignment number 5
Transform image coding

Deadline: May, 31

Libraries `scipy`, `numpy` and `matplotlib` can be useful for this assignment.

The objective of this assignment is to implement lossy image compression using the most important and commonly used method of this kind: transform coding. You have to program a function that does the lossy compression transformation on an image and returns the decompressed image, with varying parameters, to see how the change of parameters affects the image quality. Here by a "matrix" we mean a rectangular array of numbers.

The function has to be named `lossy_transform` and has three parameters:

- an image `img`, given as a matrix of $m \times n$ bytes (typically 512×512 but it can be any size): coefficients of the matrix are integers in the range 0–255. This matrix produces an image using the appropriate function of `Python`.
- a transform matrix `T`, given as a square $N \times N$ invertible matrix of floating point numbers (typically $N = 8$ but other sizes are also accepted).
- a quantization square matrix `Q`, of the same size $N \times N$ than the transform matrix `T`, with nonzero positive integer entries.

The function must first compress the image using the given transform and quantization matrices:

1. The image is split into $N \times N$ blocks B_s. If the number of rows or columns are not multiple of N then extra rows or columns are appended at the end containing copies of the last row or column; after decompression they are removed.
2. Each block B_s of pixels, representing a square region of the image in the spatial domain, is transformed to the frequency domain by computing $B_f = T \cdot B_s \cdot T^t$.
3. The frequency coefficients are the entries of B_f. They are quantized dividing each coefficient by the corresponding entry of the quantization matrix Q and rounding to the nearest integer. The result is a matrix B_q, which contains quantized versions of the frequency coordinates of the block. In this step some information is irretrievably lost.

Now the function applies the inverse transform to recover the image:

1. The coefficients of the matrix B_q are multiplied by the coefficients of the quantization matrix Q giving as a result an approximation B'_f of the frequency coefficients matrix B_f.

2. An approximation B'_s of the spatial coefficients is recovered by applying the inverse transform $B'_s = T^{-1} \cdot B'_f \cdot (T^t)^{-1}$, with entries rounded to the nearest integer so that the entries of the matrix B'_s are bytes.

3. The blocks B'_s are put together to reconstruct the shape of the original image and the appended rows and columns are removed if necessary.

The function must return the recovered image `newimg` after compression and decompression: it is a matrix of bytes of the same size than the input image `img`.

Experimentation. Use your implementation to experiment with images. You may find standard images used for testing of compression methods in

`http://sipi.usc.edu/database/database.php`)

and with different transform matrices (Hadamard, cosine, Haar, identity, etc.) and sizes (8, full image size, etc.), and also with different quantization matrices (standard JPEG; see for example:

`https://en.wikipedia.org/wiki/Quantization_(image_processing)`,

all entries of `Q` with the same constant value, etc.) and see the effect on the image.

You can visualize the error of the decompressed image with respect to the original by plotting the matrix of the absolute values of the difference `img`$-$`newimg`. In order to better see this "error image" you may have to magnify the error values multiplying by some number.