Homework 1

Meindi Zahiri Introduction to Signal and Image Processing

March 24, 2021

1 Uniform Quantization and Audio

1.1 Uniform_quantization function

I based myself on the code from this website

```
def uniform_quantization(inputSig, min_amplitude, max_amplitude,
        quantization_levels):
    to_values = np.linspace(min_amplitude, max_amplitude,
        quantization_levels)
    outputSig = [0] * inputSig.size
    for i in  range(inputSig.size):
        best_match = None
        best_match_diff = None
        for other_val in to_values:
            diff =  abs(other_val - inputSig[i])
            if best_match is None or diff < best_match_diff:
                best_match = other_val
                best_match_diff = diff
            outputSig[i] = best_match
    return outputSig
```

The function compares a value from the signal with each possible values (quantization_levels) and gives the closest one by computing the error and selecting the quantization level with the smallest error. This function works but is not efficient at all because it iterates through each values from the array quantization_levels.

1.2 Testing uniform_quantization function

1.2.1 Generating the sine wave

To generate the sine wave, the following formula was used:

$$f(t) = A \times sin(2\pi ft + \phi) \quad (1)$$

where:

A = amplitude of the sine
f = frequency
ϕ = phase shift

1.2.2 Quantization of the sine wave

I used a frequency of 0.2Hz (compared to 1Hz as asked) to be able to have just one period and be able to see better the different quantization levels.

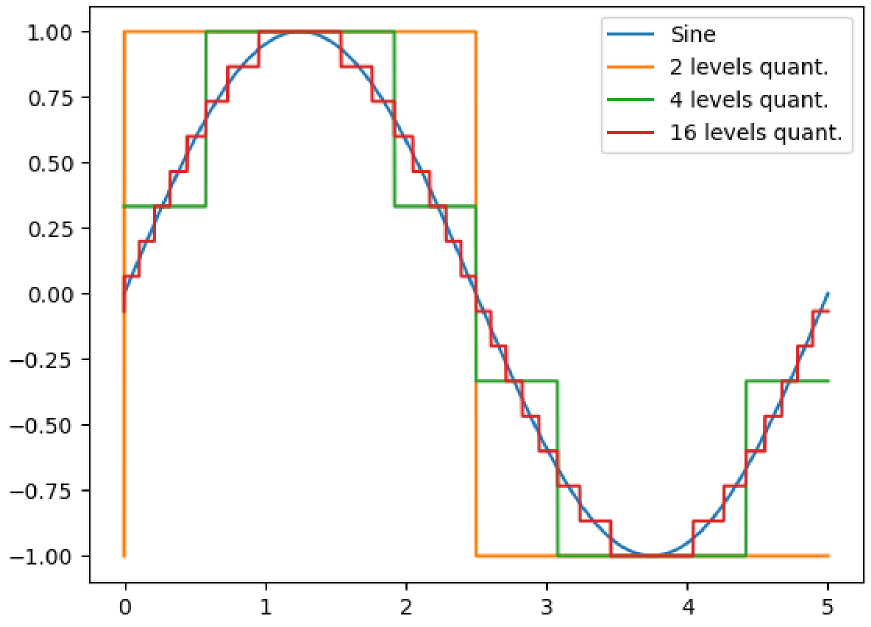

Figure 1: Sine wave plot with its 3 quantization levels

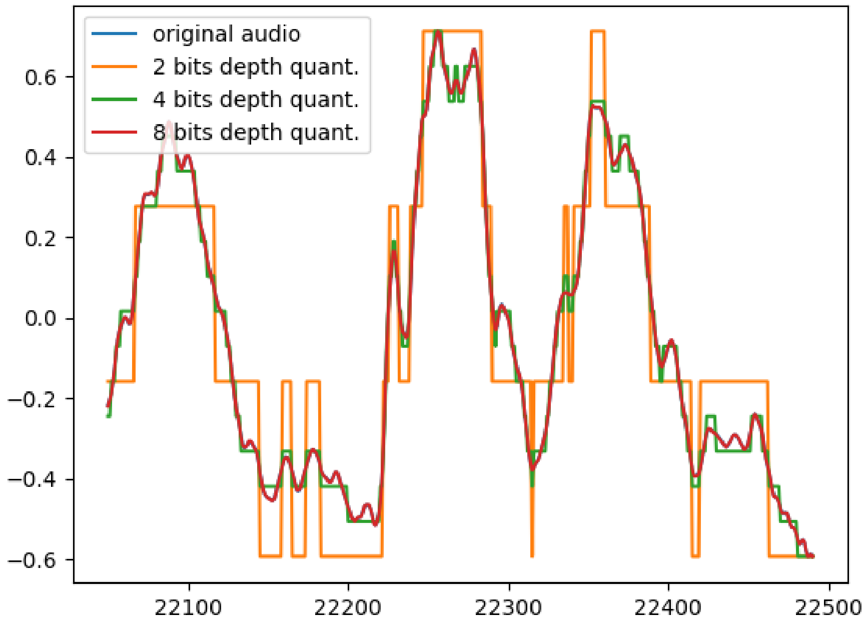

Figure 2: original audio plot (between 0.5 and 0.51s) with its 3 quantization levels

1.3 quantization of a sound file

The same quantization function was used on the audio file supplied with the assignment

i couldn't do the 16 bit depth because my quantization function is not efficient, the highest i could do to get a reasonable amount of time was 8 bits depth.

We can hear the quantization noise in the 2 and 4 bits depth quantization. The 8 bits depth version sounds exactly as the original version (in my opinion).

2 Lloyd-Max quantization

2.1 Obtaining z_k

$$\epsilon = \sum_{k=1}^{K} \int_{z_k}^{z_{k+1}} (z - q_k)^2 p(z)\, dz \tag{2}$$

by minimizing equation 1 with respect to z_k, we need to solve:

$$\frac{d}{dz_k}\epsilon = 0 \tag{3}$$

we get:

$$\begin{aligned} \frac{d}{dz_k}\epsilon &= 0 \\ \Rightarrow \frac{d}{dz_k}\sum_{k=1}^{K} \int_{z_k}^{z_{k+1}} (z - q_k)^2 p(z)\, dz &= 0 \end{aligned} \tag{4}$$

Only two components of the sum contains z_k the others don't, so they would disappear from derivating according to z_k. We then get:

$$\begin{aligned} \frac{d}{dz_k}\epsilon &= 0 \\ \Rightarrow \frac{d}{dz_k}\sum_{k=1}^{K} \int_{z_{k-1}}^{z_k} (z - q_{k-1})^2 p(z)\, dz + \frac{d}{dz_k}\sum_{k=1}^{K} \int_{z_k}^{z_{k+1}} (z - q_k)^2 p(z)\, dz &= 0 \end{aligned} \tag{5}$$

By using the Leibniz rule of integral we get:

$$\begin{aligned} (z_k - q_{k-1})^2 - (z_k - q_k)^2 &= 0 \\ z_k^2 - 2z_k q_{k-1} + {q_{k-1}}^2 - (z_k^2 - 2z_k q_k + {q_k}^2) &= 0 \\ z_k &= \frac{{q_k}^2 - {q_{k-1}}^2}{2q_k - 2q_{k-1}} \\ z_k &= \frac{(q_k + q_{k-1})(q_k - q_{k-1})}{2q_k - 2q_{k-1}} \\ z_k &= \frac{q_k + q_{k-1}}{2} \end{aligned} \tag{6}$$

2.2 Obtaining q_k

$$\epsilon = \sum_{k=1}^{K} \int_{z_k}^{z_{k+1}} (z - q_k)^2 p(z)\, dz \tag{7}$$

by minimizing equation 1 with respect to q_k, we need to solve:

$$\frac{d}{dq_k}\epsilon = 0 \tag{8}$$

we get:

$$\begin{aligned} \frac{d}{dq_k}\epsilon = 0 \\ \Rightarrow \frac{d}{dq_k}\sum_{k=1}^{K}\int_{z_k}^{z_{k+1}}(z-q_k)^2 p(z)\,dz = 0 \end{aligned} \tag{9}$$

Only one component of the sum contains q_k the others don't, so they would disappear from derivating according to q_k. We then get:

$$\frac{d}{dq_k}\int_{z_k}^{z_{k+1}}(z-q_k)^2 p(z)\,dz = 0 \tag{10}$$

By using the Leibniz rule of integral again, we get:

$$\begin{aligned} \frac{d}{dq_k}\int_{z_k}^{z_{k+1}}(z-q_k)^2 p(z)\,dz &= \int_{z_k}^{z_{k+1}}\frac{\partial}{\partial q_k}(z-q_k)^2 p(z)\,dz \\ &= \int_{z_k}^{z_{k+1}}(2z-2q_k)p(z)dz \\ \Rightarrow 2\int_{z_k}^{z_{k+1}} zp(z)dz &= 2\int_{z_k}^{z_{k+1}} q_k p(z)dz \\ \Rightarrow \int_{z_k}^{z_{k+1}} zp(z)dz &= \int_{z_k}^{z_{k+1}} q_k p(z)dz \end{aligned} \tag{11}$$

q_k is constant so we can put it outside of the integral:

$$\begin{aligned} \int_{z_k}^{z_{k+1}} zp(z)dz &= q_k\int_{z_k}^{z_{k+1}} p(z)dz \\ \Rightarrow q_k &= \frac{\int_{z_k}^{z_{k+1}} zp(z)dz}{\int_{z_k}^{z_{k+1}} p(z)dz} \end{aligned} \tag{12}$$

3 Bilinear interpolation

3.1 Linear interpolation

3.1.1 Linear interpolation function

The interpolation function is based from the formula from the lecture:

$$y = y_0 + \frac{x-x_0}{x_1-x_0}\cdot(y_1-y_0) \tag{13}$$

what was added to the function is that if the requested x point is smaller than the first x value (extrapolation) then the result would be the same value as the first x value. This was necessary in order to pass the test function.

3.1.2 Testing the linear interpolation function and 1D rescaling

The functions test_interp and test_interp_1D (supplied with assignment) were used to test the linear interpolation function.

3.2 Bilinear interpolation

3.2.1 2D Rescaling

To do the bilinear interpolation, the function linear interpolation was used twice, once for each axis

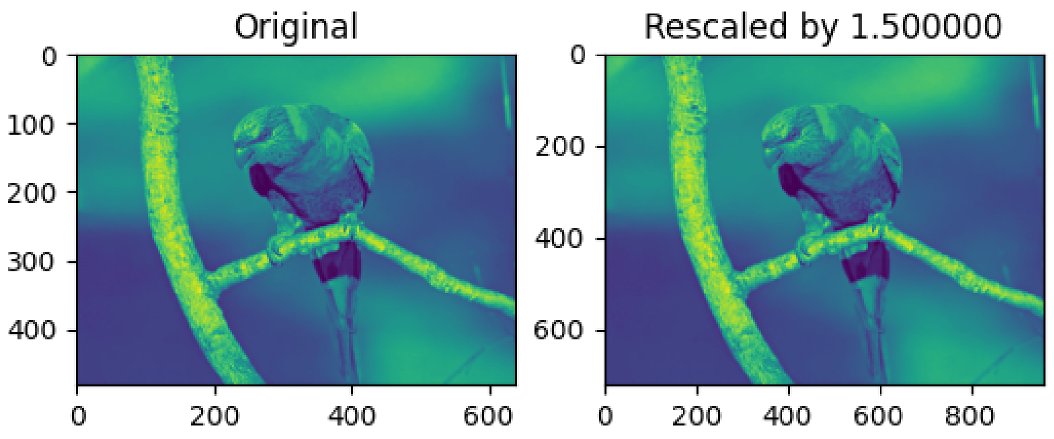

Figure 3: Bilinear interpolation applied to a greyscale image

We can see from the axis values that the image was indeed rescaled.

3.2.2 2D Rescaling (RGB image)

To apply the bilinear interpolation to a rgb image, the same function as the one used in the 2D rescaling was used, but 3 times, once for each channel

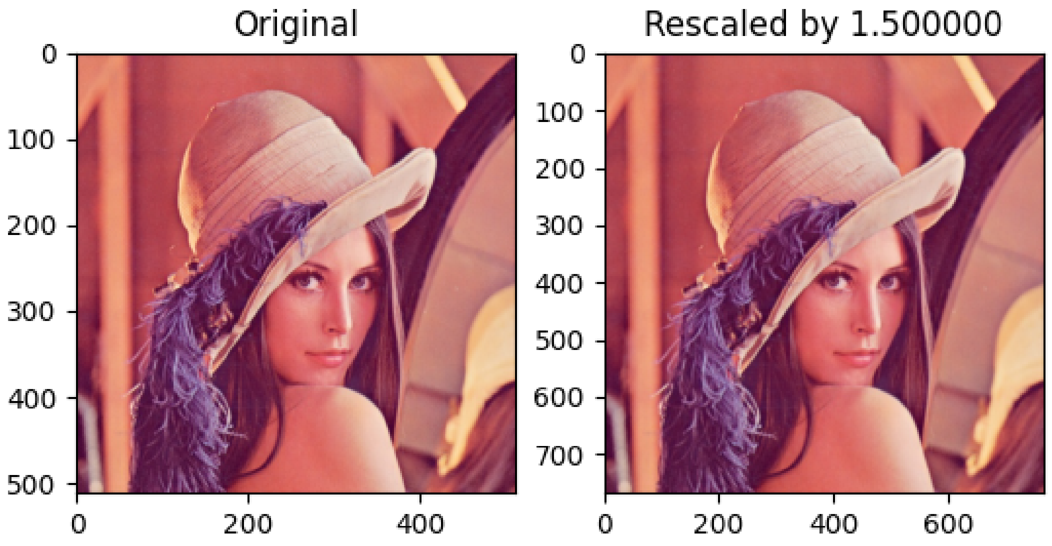

Figure 4: Bilinear interpolation applied to a rgb image